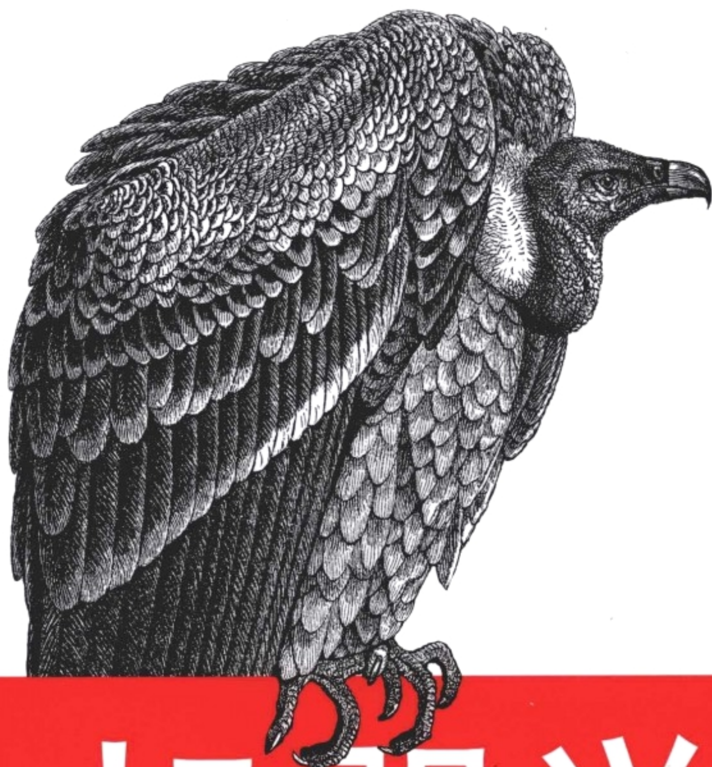

HZ BOOKS
华章科技

Machine Learning for Hackers



机器学习

实用案例解析

Drew Conway & John Myles White 著
陈开江 刘逸哲 孟晓楠 译
罗森林 审校

O'REILLY®


机械工业出版社
China Machine Press

Programming/Machine Learning

机器学习：实用案例解析

机器学习是计算机科学和人工智能中非常重要的一个研究领域，近年来，机器学习不但在计算机科学的众多领域中大显身手，而且成为一些交叉学科的重要支撑技术。本书比较全面系统地介绍了机器学习的方法和技术，不仅详细阐述了许多经典的学习方法，而且讨论了一些有生命力的新理论、新方法。

全书案例既有分类问题，也有回归问题；既包含监督学习，也涵盖无监督学习。本书讨论的案例涉及分类、回归、聚类、降维、最优化问题等。这些案例包括：垃圾邮件识别、智能收件箱、预测网页访问量、文本回归、密码破译、构建股票市场指数、用投票记录对美国参议员聚类、给用户推荐R语言包、分析社交图谱、给问题找到最佳算法等。各章对原理的叙述力求概念清晰、表达准确，突出理论联系实际，富有启发性，易于理解。在探索这些案例的过程中用到的基本工具就是R编程语言。

本书主要内容：

- **开发一个朴素贝叶斯分类器，仅仅根据邮件的文本信息来判断邮件是否是垃圾邮件；**
- **使用线性回归来预测互联网排名前1000网站的PV；**
- **利用文本回归理解图书中词与词之间的关系；**
- **通过尝试破译一个简单的密码来学习优化技术；**
- **利用无监督学习构建股票市场指数，用于衡量整体市场行情；**
- **根据美国参议院的投票情况，从统计学的角度对美国参议员聚类；**
- **通过k近邻算法向用户推荐R语言包；**
- **利用Twitter数据构建一个“你可能感兴趣的人”的推荐系统；**
- **模型比较：给问题找到最佳算法。**

“这本书为机器学习技术提供了一些非常棒的案例研究。它并不是一本关于机器学习的工具书或者理论书籍，而是对学习过程的指南，因而适合任何具有编程背景和定量思维的人。”

——Max Shron
OkCupid

Drew Conway 机器学习专家，拥有丰富的数据分析、处理工作经验。目前主要利用数学、统计学和计算机技术研究国际关系、冲突和恐怖主义等。他拥有纽约大学博士学位，曾为多种杂志撰写文章，是机器学习领域的著名学者。

John Myles White 机器学习专家，拥有丰富的数据分析、处理工作经验。目前主要从理论和实验的角度来研究人类如何做出决定，同时还是ProjectTem[illegible]log4r等流行R语言程序包[illegible]维护者。他拥有普林斯顿大学士学位，发表过许多关于机器[illegible]习的论文，并在众多国际会议发表演讲。



O'REILLY®
oreilly.com.cn

O'Reilly Media, Inc.授权机械工业出版社出版

客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com
华章网站：www.hzbook.com
网上购书：www.china-pub.com

上架指导：计算机/程序设计
ISBN 978-7-111-41731-6



定价：69.00元

机器学习：实用案例解析

Drew Conway & John Myles White 著

陈开江 刘逸哲 孟晓楠 译

罗森林 审校

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权机械工业出版社出版

机械工业出版社

图书在版编目（CIP）数据

机器学习：实用案例解析/（美）康威（Conway, D.）等著；陈开江，刘逸哲，孟晓楠译. —北京：机械工业出版社，2013.3

（O'Reilly精品图书系列）

书名原文：Machine Learning for Hackers

ISBN 978-7-111-41731-6

I. 机… II. ①康… ②陈… ③刘… ④孟… III.机器学习 IV. TP181

中国版本图书馆CIP数据核字（2013）第042873号

北京市版权局著作权合同登记

图字：01-2012-4851号

书　　名/ 机器学习：实用案例解析
书　　号/ ISBN 978-7-111-41731-6
责任编辑/ 秦健
封面设计/ Karen Montgomery，张健
出版发行/ 机械工业出版社
地　　址/ 北京市西城区百万庄大街22号（邮政编码 100037）
印　　刷/ 藁城市京瑞印刷有限公司印刷
开　　本/ 178毫米×233毫米　16开本　20印张（含1印张彩插）
版　　次/ 2013年4月第1版　2013年4月第1次印刷
定　　价/ 69.00元（册）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：(010)88378991　88361066
购书热线：(010)68326294　88379649　68995259
投稿热线：(010)88379604
读者信箱：hzjsj@hzbook.com

O’Reilly Media, Inc.介绍

O’Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O’Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O’Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar博客有口皆碑。”

——Wired

“O’Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O’Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O’Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

译者序

当今各行业，尤其是互联网，数据规模越来越大，要从中有效地发现模式来提高生产力，用传统的方式已经几乎不可能，只能借助计算机来完成诸多使命。因此，机器学习这一新兴的学科变得越来越重要，它已经在搜索、推荐、数据挖掘等多个领域闪耀光芒。机器学习是一门交叉学科，内容涉及概率论、统计学、高等数学、计算机科学等多门学科。该学科致力于设计一种让计算机具有“学习”能力的算法，通过发现经验数据中隐藏的模式，实现对未知数据的预测。

大数据时代是机器学习最美好的时代，因为数据不再是问题，各类问题都可以收集到海量的数据。但是，对于很多人来说，这一门交叉学科本身却神秘而陌生，对于没有系统学习过相关基础学科的人来说尤其感到“高不可攀”。如今已出版的机器学习相关书籍中，很多都有这个特点：公式多，晦涩难懂。这让很多程序员出身的人望而却步。然而，在第一次读到本书的英文版时，译者就彻底相信：机器学习完全可以讲解得通俗易懂，让知识的传递实现“润物细无声”。

本书秉承的原则是：实践出真知，只要多动手，没有攻克不了的技术难题。因此作者预期的阅读对象是如电脑黑客般的人，要求对技术有发自内心的求知欲和好奇心，愿意自己动手而非纸上谈兵。全书精心选择了12个机器学习案例，由浅入深，面面俱到，既有基础知识（如数据分析），也有当前热门的社交网站推荐案例。书中的每一个案例都由作者娓娓道来，逐一剖析关键算法的代码，没有丝毫学究气息，触动每个机器学习初学者的内心最深处。

书中所有算法都采用R语言实现。R语言是一门用于统计学的开源脚本语言，基于它的开源性，有来自世界各地的开源拥护者贡献的各种统计学相关的程序包，稳定且方便，尤其是它对数据可视化的支持，更是一柄利器，既轻巧又实用。书中所有源代码和数据在

原书的官方网站上都可以免费下载。在阅读过程中，犹如作者亲至身侧，为你讲解代码和思路，为你排除错误和优化效果。

全书案例既有分类问题，也有回归问题；既包含监督学习，也涵盖无监督学习。所选择的案例妙趣横生，如分析UFO目击记录、破译密码、预测股票、分析美国参议员“结党”的情况，等等，这里就不“剧透”了，大家自己去享受学习的乐趣吧。

书中12个案例之间的依赖关系不是特别强（除R语言基础知识外，其余某几章仅有个别知识点之间存在依赖性），可以像连续剧一样，逐一播放，也可以像一个个小品一般，挑感兴趣的内容分别播放。学习完这些案例之后，相信你会窥见机器学习的一斑，然后再根据自己的实际情况更深入地学习。

本书翻译工作由三位来自互联网世界的工程师通力协作完成，其中，来自新浪微博的陈开江负责完成前言及第1～4章的翻译；来自阿里B2B的刘逸哲负责完成第5、8、9和11章的翻译；来自阿里一淘的孟晓楠负责完成第6、7、10和12章的翻译；同时，全书审校工作由来自北京理工大学的罗森林教授义务承担。

本书能够得以出版，首先要感谢机械工业出版社的吴怡编辑，是她给了我们三位工程师这个学习知识并传递知识的机会，她经验丰富，在翻译过程中给予了我们许多建设性的指导意见。其次，要感谢罗森林教授，他在百忙之中为我们担任全书的审校工作，从而让国内的机器学习者能感受到这本书应有的魅力。最后，我们要感谢互联网，因为译者与本书的缘分始于互联网，从看到原书、报名翻译、组成翻译团队、翻译过程中的讨论，所有这样都是通过互联网完成的。

虽然经过罗森林教授认真审校并且给我们提出了宝贵意见，但是由于译者本身水平有限，书中译文势必还存在不妥甚至错误之处，恳请机器学习界的广大前辈、同仁们不吝赐教，促使我们继续为大家更好地传递先进技术，让更多机器学习爱好者成为机器学习的黑客。

我们坚信集体智慧是再高的个人智慧都无法企及的，因此真诚希望大家一起来贡献自己的智慧。三位译者的微博分别为：*http://weibo.com/kaijiangidan*（陈开江，@刑无刀）、*http://weibo.com/liuyizhe10*（刘逸哲，@刘逸哲）、*http://weibo.com/u/1911115643*（孟晓楠，@*XiaonanMeng*）。无论是对翻译本身有任何意见或建议，还是对机器学习方面有心得，都欢迎大家到我们的微博上交流、切磋，我们一起贡献自己的智慧，在集体智慧中互相学习，共同进步。

作者介绍

Drew Conway 机器学习专家，拥有丰富的数据分析与处理工作经验。目前主要利用数学、统计学和计算机技术研究国际关系、冲突和恐怖主义等。他曾作为研究员在美国情报和国防部门供职数年。他拥有纽约大学政治系博士学位，曾为多种杂志撰写文章，是机器学习领域的著名学者。

John Myles White 机器学习专家，拥有丰富的数据分析与处理工作经验。目前主要从理论和实验的角度来研究人类如何做出决定，同时还是几个流行的R语言程序包的主要维护者，包括ProjectTemplate和log4r。他拥有普林斯顿大学哲学系博士学位，曾为多家技术杂志撰稿，发表过许多关于机器学习的论文，并在众多国际会议上发表演讲。

译者介绍

罗森林 博士，教授，博导。现任北京理工大学信息系统及安全对抗实验中心主任、专业责任教授。国防科技工业局科学技术委员会成员；《中国医学影像技术杂志》、《中国介入影像与治疗学》编委会委员；全国大学生信息安全技术专题邀请赛专家组副组长；中国人工智能学会智能信息安全专业委员会委员等。主要研究方向为信息安全、数据挖掘、媒体计算、中文信息处理等。负责或参加完成国家自然科学基金、国家科技支撑计划、863计划、国家242计划等省部级以上项目40余项。已发表学术论文90余篇，出版著作8部，出版译著1部，获授权专利3项。

陈开江 新浪微博搜索部研发工程师，曾独立负责微博内容反垃圾系统、微博精选内容挖掘算法、自助客服系统（包括自动回复、主动挖掘、舆情监测）等项目，目前主要从事社交挖掘、推荐算法研究、机器学习、自然语言处理相关工作，研究兴趣是社交网络的个性化推荐。

刘逸哲 阿里巴巴，CBU基础平台部搜索与推荐团队核心技术与query分析方向负责人，机器学习技术领域及圈子负责人。曾任中国雅虎相关性团队、自然语言处理团队算法工程师；AvePoint.inc开发工程师，从事企业级搜索引擎开发。研究兴趣是机器学习、自然语言处理及个性化推荐等算法在大规模数据上的应用。

孟晓楠 一淘广告技术，阿里非搜索广告算法负责人，负责用户行为分析、建模与细分，RTB竞价算法，展示广告CTR预估与SEM优化。曾工作于网易杭州研究院，参与过分布式全文检索系统和网易博客产品的数据挖掘算法开发。研究兴趣是计算广告技术、机器学习、大数据技术、信息检索等。

封面介绍

本书封面动物是兀鹫（griffon vulture，鹰科）。这种庞然大鸟分布在旧大陆（即欧、亚、非）较暖和的地区，也就是说地中海附近。

这类鸟头部的羽毛呈白色且稀少，翅膀宽大，尾巴短小。成年兀鹫——身高在0.9~1.1m、翅宽平均在2.3~2.8m——通常身体羽毛呈黄棕色，间杂黑色，颈部周围羽毛呈白色。兀鹫是一种食腐动物，只捕食死尸。

兀鹫最长寿命现存记录是41.4年（养殖场记录）。它们广泛分布在欧洲南部、非洲北部山区，以及亚洲。每次产蛋仅一枚。



推荐阅读



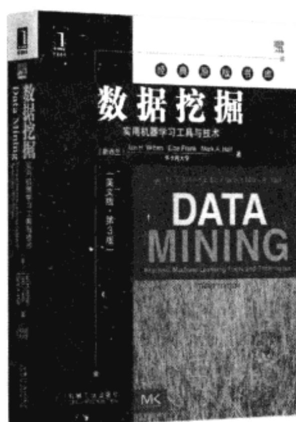
计算机与机器视觉：理论、算法与实践

作者：E. R. Davies ISBN：978-7-111-41232-8 定价：128.00元



数据挖掘：概念与技术

作者：Jiawei Han ISBN：978-7-111-39140-1 定价：79.00元



数据挖掘：实用机器学习工具与技术

作者：Ian H. Witten等 ISBN：978-7-111-37417-6 定价：108.00元



算法精解：C语言描述

作者：Kyle Loudon ISBN：978-7-111-39426-6 定价：79.00元

推荐阅读

HTML 5应用开发实践指南

作者：Zachary Kessin ISBN：978-7-111-41451-3 定价：49.00元

机器学习：实用案例解析

作者：Drew Conway 等 ISBN：978-7-111-41731-6 定价：69.00元

Visual C++并行编程实战

作者：Colin Campbell 等 ISBN：978-7-111-38806-7 定价：59.00元

PHP精粹：编写高效PHP代码

作者：Davey Shafik 等 ISBN：978-7-111-39907-0 定价：59.00元

程序员度量：改善软件团队的分析学

作者：Jonathan Alexander ISBN：978-7-111-40140-7 定价：59.00元

编写可读代码的艺术

作者：Dustin Boswell 等 ISBN：978-7-111-38544-8 定价：59.00元

Android程序设计

作者：Zigurd Mednieks ISBN：978-7-111-40184-1 定价：79.00元

Android 应用开发攻略

作者：Ian F. Darwin ISBN：978-7-111-41411-7 定价：99.00元

精通搜索分析

作者：Brent Chaters ISBN：978-7-111-39681-9 定价：69.00元



目录

前言

致机器学习的黑客们

为了更好地阐释本书的切入点，很有必要对“机器学习”与“黑客”这两个词语下个定义。

什么是机器学习？简单来说，机器学习就是一套工具和方法，凭借这些工具和方法我们可以从观测到的样本中提炼模式、归纳知识。举个例子，如果我们要让计算机识别信封上的邮政编码，那么需要这样的数据：首先是信封的图片数据，其次信封上必须有收件人的邮政编码。换句话说，在特定情境下，我们可以记录研究对象的行为，从中学习，然后对其行为建模，该模型反过来促进我们对该情境有更深入的理解。在实际项目中，机器学习需要数据，而且对当今的应用程序来说不是一点点数据（有可能达TB级的数据）。大多数机器学习技术不担心没有数据，现代企业运营所产生的数据量之大意味着这些技术应用的春天来了。

什么是黑客呢？在我们眼中，黑客就是喜好用新技术进行实验、解决问题的人，而与“网络罪犯”、“不法少年”这些世俗字眼完全无关。如果你曾经手捧O'Re illy最新出版的一本关于一门新计算机语言的图书，跌跌撞撞地敲下代码，并最终调试通过了你的第一个程序，那么你就称得上是一名黑客。或者，你曾把新买来的小机器大卸八块，并最终弄懂了它的整个机械结构，那么你也是个黑客。通常，黑客这样做并无特别的原因，只是为了享受这个过程，只是为了要彻底了解一门未知的技术。

计算机黑客对事物原理有一种与生俱来的好奇心和动手的热情，他们（与之相对应的还有汽车黑客、生活黑客、美食黑客，等等）还有软件设计和开发的经验，他们就是以前写过程序的人，甚至很可能使用过很多种语言。对于一个黑客来说，UNIX不是一个四

个字母的单词，工作时用命令行导航和bash shell操作与用图形用户界面一样熟练。处理文本时，黑客首先想到的就是正则表达式，以及sed、awk、grep这些工具。在本书的写作中，我们也假设读者在这些方面的知识水平比较高。

本书的组织结构

机器学习融合了许多传统领域的理论和技术，诸如数学、统计学和计算机科学等。因此，学习本学科存在许多切入点。由于数学和统计学是机器学习的理论基础，因此新手应该在一定程度上掌握机器学习基础技术的范式。市面上已有很多这方面的优秀书籍，如Hastie、Tibshirani、Friedman三人的经典著作《统计学习基础》（The Elements of Statistical Learning，[HTF09]，完整信息见参考书目）[注1]。但是在黑客们的人生理念里，很重要的一部分就是：边做边学。很多黑客在面对问题的时候，更习惯于在实际操作过程中寻找解决方案，而非从理论基础出发推导解决方案。

从这个角度而言，教授机器学习的另一种方法就是采用案例式教学。例如，在讲解推荐系统时，我们会提供一批训练样本和一版模型，然后看看模型如何使用训练样本。类似的参考书有很多，比较新的一本是Segaran的《集体智慧编程》（Programming Collective Intelligence，[Seg07]）。以上的讨论只是介绍了操作方式，却没有解释为什么这样做。在理解了一个方法的原理时，我们也许还想知道为何这个方法适用于某个情境，或者为何解决了某个特定的问题。

因此，为了给机器学习的黑客们提供一本更全面的参考书，我们必须在学科理论的深度和应用探索的广度之间寻求一个平衡。为了达到这个目的，我们决定采用案例教学的方式来教授机器学习。

最好的学习方法是：首先带着问题思考，然后专心研究解决问题的方法。这也是案例教学能有效执行的机理所在。不同之处在于，我们并不是拿一些还没有成熟解决方法的机器学习问题来举例，而是讨论一些已深入理解和广泛研究的问题，并列举了一些特定的案例辅以说明。对于这些案例，有些方法可以很好地解决问题，而有些方法却根本不适用。

基于上述指导思想，本书的每一章都是基于特定机器学习问题的独立案例研究。本书前几个案例从分类讲到回归（第1章会进一步讨论），然后又讨论了聚类、降维、最优化问题等。需要说明的是，不是所有的问题都可以简单地归为分类问题或者回归问题，书中涉及的一些案例同时包含分类与回归问题（有些比较明显，有些不易察觉）。以下是本书中出现的所有案例研究的简要介绍（按出现先后顺序）。

注1： 这本书也可以从*http://www-stat.stanford.edu/~tibs/ElemStatLearn/*免费下载。

文本分类：垃圾邮件识别

这一章介绍由电子邮件文本数据引起的二分类问题。在此要处理的是机器学习中的经典问题：将某个输入识别为两个类中的一个，在这里指的就是正常邮件（合法的电子邮件）或垃圾邮件（用户不希望看到的邮件）。

项目排序：智能收件箱

与上个案例一样，这里还是采用电子邮件文本数据，但是不再研究二分类问题，而是上升到研究一组具体的类别。具体来说，就是要在某一电子邮件中识别并抽取适当的特征，这些特征使该邮件在所有邮件中处于优先阅读的位置。

回归模型：预测网页访问量

现在介绍机器学习的第二个基本工具——线性回归。这里要处理的是关系大致逼近一条直线的数据。在这个案例研究中，目的是预测互联网上排名前1000（2011年）的网页访问量。

正则化：文本回归

有时候，我们并不能用一条直线很好地描述数据间的关系。为了描述这个关系就要用另一种函数来拟合，但同时又要防止出现过拟合。正则化的方法可以克服这一问题，同时通过一个案例加以说明，主要目的是理解O’Reilly图书中词与词之间的关系。

最优化：密码破解

机器学习中几乎每一个算法都可以看成是最优化问题，比如，将预测错误率最小化。这里绍一个经典的算法来实现最优化，并尝试用这个算法破解一段字母密码。

无监督学习：构建股票市场指数

到目前为止我们的讨论还局限于有监督的学习技术。在此要介绍机器学习方法上的另一半：无监督学习。两者最主要的不同是：有监督学习方法是使用结构化数据进行预测，而无监督学习是为了在数据中发现结构。因此，我们将用一批股票市场的数据来构建一个指数，这个指数可以衡量整体市场行情的好坏。

空间相似度：用投票记录对美国参议员聚类

这里介绍这样一个概念：样本点的空间距离。为了实现对参议员聚类，需要设计距离的测算方法，以及样本点基于空间距离的聚类方法。我们用美国参议员记名投票的数据，根据其所得投票记录对这些立法者进行聚类。

推荐系统：给用户推荐R语言包

为深入讨论空间相似度，我们将讨论如何搭建一个基于样本空间密度的推荐系统。这一章介绍K近邻算法，并根据程序员安装的R语言函数包，用这个算法来给他们推荐其他的R语言包。

社交网络分析：在Twitter上感兴趣的人

　　这一章会结合之前讨论过的许多概念，并引入一些新的概念与方法，用Twitter数据设计并搭建一个“可能感兴趣的人”的推荐系统。在这个例子中，将搭建一个系统用于下载Twitter数据，发现其中的圈子，然后用基本的社交网络分析技术向用户推荐可能感兴趣的人。

模型比较：给你的问题找到最佳算法

　　最后一章讨论的是用于选择解决问题的最佳机器学习方法的技巧。这一章将介绍最后一个算法——支持向量机，并采用在第3章中介绍的垃圾邮件数据来比较其与其他算法的优劣。

我们在探索这些案例的过程中用到的基本工具就是R统计编程语言（*http://www.r-project.org/*）。R语言非常适合于机器学习的案例研究，因为它是一种用于数据分析的高水平、功能性脚本语言。很多基础算法框架已经内置在R语言中，或者已经在一些R语言包中实现了，这些包可以在综合R档案网（Comprehensive R Archive Network，CRAN）[注2]上找到。这可以避免为每一个实际项目写基础功能代码，从而把我们从重复劳动中解放出来，把精力放在思考问题的本身上。

本书约定

本书使用了以下排版约定：

斜体

　　用于新术语、URL、电子邮件地址、文件名与文件扩展名。

等宽字体（`Constant width`）

　　用于表明程序清单，以及在段落中引用的程序中的元素，如变量、函数名、数据库、数据类型、环境变量、语句、关键字。

等宽粗体（**`Constant width bold`**）

　　用于表明命令，或者需要读者逐字输入的文本内容。

等宽斜体（*`Constant width italic`*）

　　用于表示需要使用用户提供的值或者由上下文决定的值来替代的文本内容。

注意： 这个图标代表一个技巧、建议或一般性说明。

注2：　关于CRAN的更多信息，请浏览*http://cran.r-project.org/*。

警告：这个图标代表一个警告或注意事项。

示例代码的使用

本书提供代码的目的是帮你快速完成工作。一般情况下，你可以在你的程序或文档中使用本书中的代码，而不必取得我们的许可，除非你想复制书中很大一部分代码。比方说，你在编写程序时，用到了本书中的几个代码片段，这不必得到我们的许可。但若将O'Reilly图书中的代码制作成光盘并进行出售或传播，则需获得我们的许可。引用示例代码或书中内容来解答问题无需许可。将书中很大一部分的示例代码用于你个人的产品文档，这需要我们的许可。

如果你引用了本书的内容并标明版权归属声明，我们对此表示感谢，但这也不是必需的。版权归属声明通常包括：标题、作者、出版社和ISBN号，例如："*Machine Learning for Hackers* by Drew Conway and John Myles White (O'Reilly). Copyright 2012 Drew Conway and John Myles White, 978-1-449-30371-6"。

如果你认为你对示例代码的使用已经超出上述范围，或者你对是否需要获得示例代码的授权还不清楚，请随时联系我们：*permissions@oreilly.com*。

联系我们

有关本书的任何建议和疑问，可以通过下列方式与我们取得联系：

美国：

> O'Reilly Media，Inc.
> 1005 Gravenstein Highway North
> Sebastopol，CA 95472

中国：

> 北京市西城区西直门南大街2号成铭大厦C座807室（100035）
> 奥莱利技术咨询（北京）有限公司

我们会在本书的网页中列出勘误表、示例和其他信息。可以通过*http://shop.oreilly.com/product/0636920018483.do*访问该页面。

要评论或询问本书的技术问题，请发送电子邮件到：

> *bookquestions@oreilly.com*

想了解关于O'Reilly图书、课程、会议和新闻的更多信息，请访问以下网站：

http://www.oreilly.com.cn
http://www.oreilly.com

还可以通过以下网站关注我们：

我们在Facebook上的主页：*http://facebook.com/oreilly*

我们在Twitter上的主页：*http://twitter.com/oreillymedia*

我们在Youtube上的主页：*http://www.youtube.com/oreillymedia*

致谢

两位作者的共同致谢：

首先，我们要感谢编辑Julie Steele，他为我们第一本书的问世出力不少。其次，我们要感谢Melanie Yarbrough和Genevieve d'Entremont，由于他们细致入微的校对工作，才使本书得以出版。此外，我们还要感谢O'Reilly公司给我们的书建言献策的其他幕后工作人员。除了O'Reilly的工作人员，我们还要感谢本书的技术审校者Mike Dewar、Max Shron、Matt Canning、Paul Dix和 Maxim Khesin，他们的意见让这本书有了很大的提升，本书若再有错误出现，那就是我们自己的问题了。

我们也想感谢NYC Data Brunch数据中心的成员们，最初是他们鼓励我们写这本书的，并且给我们提供场地来完善机器学习教学这一想法。我们特别要感谢Hilary Mason把我们引荐给了O'Reilly公司。

最后，我们要感谢数据科学圈内的很多朋友，他们非常支持我们，不断鼓励我们，让我们在整个写作过程中干劲十足。因为知道人们在期待这本书，所以我们在写作整本书这段漫长旅途上从未停下过脚步。

Drew Conway的致谢：

我想感谢编辑Julie Steele，她欣赏我们的写作动机，并帮助我们出版这本书。我要感谢在本书写作过程中和完成后所有给我们反馈意见的人们，特别是Mike Dewar、Max Shron、Matt Canning、Paul Dix和 Maxim Khesin。我要感谢我的妻子Kristen，她不仅给我灵感，而且一直陪伴我、鼓励我。最后，我要感谢我的搭档John，是他提出了写这样一本书的创意，并且坚信能够完成。

John Myles White的致谢：

首先我要感谢我的搭档Drew和我一起写了这本书。有人合作写一本书，会让整个过程易于掌控，而且乐在其中。此外，我要感谢我的双亲，他们总是鼓励我去探索感兴趣的任何一个领域。我也要感谢Jennifer Mitchel 和Jeffrey Achter，是他们不断激励我在本科阶段把精力放在数学上。大学的岁月刻画了我的世界观，我很感激在这个过程中得到他们的指导。我同样要感谢我的朋友Harek，因为是他不断地促我奋进、超越自我。此外，我还要感谢La Dispute乐队的音乐，陪伴我度过整个写作过程。最后，我要感谢所有在我工作时给我提供过容身之处的人，包括给我提供沙发的朋友们以及Boutique Hotel Steinerwirt 1493酒店和Linger Café咖啡店的老板，因为正是在这两个地方，我完成了本书的初稿和终稿。



第1章

使用R语言

机器学习是由软件工程、计算机科学这样的新兴学科与数学、统计学这样的传统学科交叉形成的一门学科。在本书中，我们会介绍统计学领域的一些有助于人们认识世界的工具。统计学一直在研究如何从数据中得到可解释的东西，而机器学习则关注如何将数据变成一些实用的东西。对两者做出如下对比更有助于理解“机器学习”这个术语：机器学习研究的内容是教给计算机一些知识，再让计算机利用这些知识完成其他的任务。相比之下，统计学则更倾向于开发一些工具来帮助人类认识世界，以便人类可以更加清晰地思考，从而做出更佳的决策。

在机器学习中，学习指的是采用一些算法来分析数据的基本结构，并且辨别其中的信号和噪声，从而提取出尽可能多的（或者尽可能合理的）信息的过程。在算法发现信号或者说模式之后，其余的所有东西都将被简单判断为噪声。因此，机器学习技术也称为模式识别算法。我们可以“训练”机器去学习数据是如何在特定情境中产生的，从而使用这些算法将许多有用的任务实现自动化。这就引出了训练集（training set）这一术语，它指的是构建机器学习过程所用到的数据集。观测数据、从中学习、自动化识别过程，这三个概念是机器学习的核心，同时也是本书的主线。本书的核心问题包含两个特别重要的模式：分类问题和回归问题，这两类问题在书中会不断出现，当然我们也会教给大家解决方法。

在本书中，我们假设读者具备相对较高水平的基本编程技能和算法知识。R语言一直是一门相当小众的编程语言，甚至对于许多资深程序员来说亦是如此。为了尽量让每一个读者都处于同一起点，本章将介绍一些R语言的入门基础知识。稍后还将介绍一个使用R语言来处理数据的延伸案例研究。

警告： 本章并非要完整地介绍R编程语言。况且仅用一个章节来完整地介绍R也根本不可能。相反，这一章是为了让读者做好相关知识准备，进而用R来完成机器学习任务，尤其是用R来加载、探索、清洗以及分析数据等。已有许多不错的资源专注于R基础知识，包括数据类型、算术运算概念以及最佳编码实践。本章要展示的案例研究与上述这些R基础概念都有关联。虽然我们会谈及上述每个问题，但都不会太深入。若读者有兴趣对这些知识点进行回顾，表1-3列出了可供参考的资源。

如果你此前从未接触过R及其语法，我们强烈建议你通过阅读本章来扫扫盲。R不像其他高层次脚本语言，如Python和Ruby，它的语法独特且晦涩，往往不如其他编程语言容易上手。如果你此前用过R，但不是用于机器学习，那么在进行案例研究之前也有必要花时间看看以下综述。

R与机器学习

> R是用于统计计算、绘图的语言和操作环境……R提供了丰富多样的统计功能（线性和非线性建模、经典统计学实验、时间序列分析、分类、聚类……）和绘图技术，并且具有高度可扩展性。在统计方法学研究中，S语言是一种常用的工具，而R则以开源的方式跻身其中。
>
> ——用于统计计算的R项目，*http://www.r-project.org/*

> R最大的优势是：它是由统计学家们开发的。R最大的劣势是……它是由统计学家们开发的。
>
> ——Bo Cowgill，Google公司

在处理和分析数据方面，R是一门非常强大的语言。它在数据科学界以及机器学习界的迅速流行已经使它成为分析学领域实际上的通用语言。R在数据分析界的成功源于上述引文中提到的两个因素：首先，R内置了统计学家们所需的技术；其次，R备受统计学界开源贡献者们的支持与推崇。

专门为统计计算设计的编程语言具有很多技术上的优势。正如R项目所述，这门语言提供了一座通往S的开源桥梁，而S中所包含的许多基础函数都是高度专业化的统计操作。例如，用R来执行一个基本的线性回归操作，你只需简单地把数据传入lm函数，然后函数返回一个包含了详细回归信息（回归系数、标准误差、残差等）的对象。然后，这个数据可以用plot函数进行可视化，plot函数是专用于对分析结果进行可视化的函数。

与科学计算相关的其他大众编程语言，比如Python，要实现和lm函数相同的功能，需要若干第三方库分别来完成数据表达（NumPy）、进行分析（SciPy）、将结果可视化

（matplotlib）等过程。这样复杂的分析步骤，R只需一行代码就可完成，我们将在接下来的几个章节中见识到这些。

此外，和其他科学计算环境一样，R的基本数据类型也是向量。在本质上，R语言里的所有的数据都是向量，尽管它们有不同的聚合和组织方式。尽管这种数据结构理念比较僵化，但考虑到R的应用，这种观点还算合乎逻辑。R中最常用到的数据结构就是数据框（data frame），可以把它看做R内核中一种带属性的矩阵、一种内部定义的数据表结构，或者一种关系型数据库式结构。从根本上讲，数据框就是将向量简单地按列聚合的结果，同时R为其提供一些特别的功能，这样一来，数据框就成为适用于任何形式数据的理想结构。

警告： 正因其有所长，R也有短板——R并不能很好地处理大数据。尽管已有很多人在努力解决，但这仍然是一个严重的问题。然而，对于我们将要探讨的案例研究来说，这不是个问题。我们使用的数据集相对较小，要搭建的系统也都只是原型系统或概念验证模型。这个区别很重要，因为如果你要搭建Google或Facebook那样规模的企业级机器学习系统，选择R并不合适。事实上，像Google或Facebook这些公司通常把R作为“数据沙箱”，用于处理数据以及实验新的机器学习方法。如果某个实验有了成果，那么工程师就会把R中的相关功能用更适合的语言复现出来，比如C语言。

这种实验精神也孕育出了R用户群的一种集体感。R的社会优势依赖于这个巨大且不断成长的专家圈，是他们坚持使用并不断丰富这一语言的。正如Bo Cowgill所言，统计学家强烈希望有一种计算环境能够满足他们的特殊需求，R才得以诞生。因此，很多R用户都是各自领域的专家，这包括一些完全不同的学科，比如数学、统计学、生物学、化学、物理学、心理学、经济学以及政治学等。这个专家圈用R大量的基础函数打造了许许多多的程序包。在本书写作之时，R的程序包资源库CRAN中就已包含2800多个程序包。在接下来的案例研究中，我们会用到许多最流行的程序包，但这也仅仅是R的皮毛而已。

虽然Cowgill的话后半句有些言重，但是这进一步强调了R用户群的力量之大。随后我们会发现，R古怪的语法常常让代码“错误百出”，这足以让许多资深程序员敬而远之。但是，在一门语言中，所有的语法难题最终都可以攻克，尤其对于那些持之以恒的黑客们而言。对于非统计学出身的人而言，更大的困难是不熟悉R中内置的数学和统计函数。比如，使用lm函数时，如果你从未接触过线性回归，你可能就不知道结果里面已经包含了回归系数、标准误差和残差，你也不知道结果该怎么解释。

因为这门语言是开源的，所以你可以随时查看函数的源代码，看看内部是怎么运行的。本书的目的之一就是要探索这些函数在机器学习的情境下怎么使用，但这最终也只能让

你窥见R所有功能的冰山一角。值得庆幸的是，在R社区中，很多人不仅乐于帮助你理解这门语言，而且乐于帮助你了解其内部的实现。表1-1列出了一些最佳的R社区。

表1-1：R的社区资源

资源	网址	介绍
RSeek	*http://rseek.org/*	当核心开发团队决定要开创一个S的开源版，并称其为R时，他们并没有考虑到在互联网上搜索与这门语言相关的资料是多么不容易，因为它只以一个字母命名。此工具专门用于缓解这一问题，提供了一个集中获得R文档与信息的渠道
Official R mailing lists	*http://www.r-project.org/mail.html*	这里有一些R语言相关的邮件列表，内容包括最新公告、程序包、开发以及帮助。许多该语言的核心开发者都经常查看该邮件列表，反馈也很简明、及时
StackOverflow	*http://stackoverflow.com/questions/tagged/r*	黑客们都知道StackOverflow.com是寻求包括R在内任何语言编程技巧的首选网络资源。得益于几位著名R用户的努力，StackOverflow上总有很多高手一直在添加和回答R相关的问题
#stats Twitter hash tag	*http://search.twitter.com/search?q=%23rstats*	在Twitter上也活跃着相当多的R用户，他们把#rstats作为他们的标签。根据这条线索你可以找到有用的资源链接、找到R高手、提问——前提是能用140个字符把问题表述清楚
R-Bloggers	*http://www.r-bloggers.com/*	上百人都在写博客分享他们如何将R用于研究、工作或只是为了好玩。R-bloggers.com把这些博客聚合起来，并且提供一个唯一链接指向所有与R相关的博客内容。这里也是用案例学习的好去处
Video Rchive	*http://www.vcasmo.com/user/drewconway*	R用户群不断壮大，关于R的地方性会议也不断增多。Rchive上传视频和演示文稿，专门记录这些会议上的演讲及专题报告，现在该网站收录的报告已覆盖了全世界的R用户

本章余下的内容专门教你如何配置好R环境并且使用它，包括下载和安装R，以及下载R程序包。本章以一个小型的案例研究作为结束，该案例研究将介绍一些我们在之后章节常常用到的R知识，包括加载、清洗、组织以及分析数据等问题。

下载和安装R

和其他开源项目一样，R有若干个不同地区的镜像下载站点。如果你的电脑上还没安装R，第一步就是要下载R。登录*http://cran.r-project.org/mirrors.html*，选择离你最近的CRAN镜像站点，选择了镜像站点后，根据你所用的操作系统下载适当的版本。

R依赖一些用C或Fortran编译的库。因此，你可以安装编译好的二进制版本，也可以选择用源代码安装，这既取决于你的操作系统，又取决于你用源代码安装软件的熟练程度。接下来我们一一演示如何在Windows、Mac OS X以及Linux环境下安装R，并配有用源代码安装和二进制安装的说明。

最后要说明一点，R既有32位版本，也有64位版本。依据硬件和操作系统的配置，你可以选择合适的版本进行安装。

Windows

网站上有两个子目录可提供Windows上的R安装文件：*base*和*contrib*。后者是一个包含了所有扩展包的Windows二进制版本，而前者仅仅是包含基本功能的二进制安装文件。选择*base*目录，然后下载最新编译的二进制文件即可进行安装。在R上安装程序包也不难，而且不受语言的限制，因此没必要用*contrib*目录下的安装文件进行安装。只要一步步跟着屏幕上的安装说明即可完成安装。

安装成功之后，在你的开始菜单中就有R应用程序的图标，可以用这些图标打开R图形用户界面（RGui）和R控制台（R Console），见图1-1。

对于Windows系统下的大多数标准安装，这个过程不会出什么问题。如果你选择了自定义安装或者在安装过程中遇到了一些问题，你可以在你所选择的镜像站点上找到R for Windows FAQ页面，查询问题。

Mac OS X

Mac OS X用户就幸运得多了，因为操作系统已经预装了R。你可以打开Terminal.app，然后在命令行中输入“R”确认是否已安装R。这样你就万事俱备了！然而，对部分用户来说，需要安装GUI应用程序来与R Console进行交互，如此就得再安装一个软件。在Mac OS X下，你也可以选择安装编译好的二进制版本，或者用源代码安装。对于没用过Linux命令行的用户，我们推荐使用二进制版本安装。只需在*http://cran.r-project.org/mirrors.html*上选择镜像站点，下载最新版本，然后按照屏幕说明操作就可以了。安装完成后，在你的Applications文件夹下有两个文件：R.app（32位版本）和R64.app（64位版本）。你可根据硬件的配置和Mac OS X的版本选择其中一个版本。

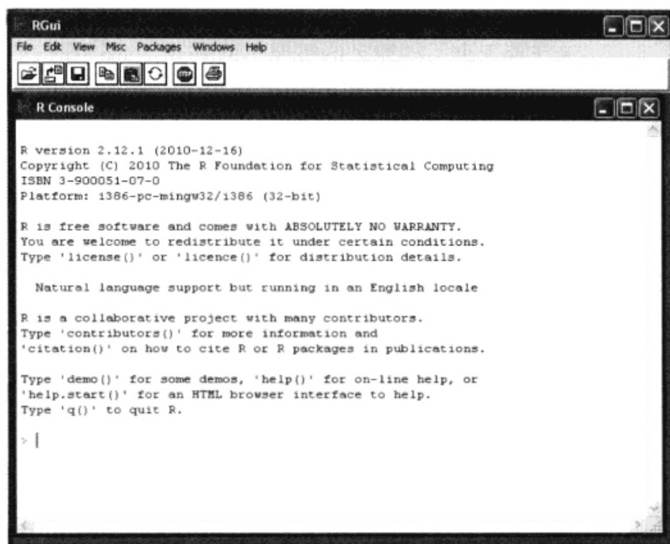


图1-1：Windows系统下RGui与R Console

与Windows系统下的安装过程一样，如果你在Mac OS X下用二进制版本安装，那么整个过程都不会有什么问题。打开刚安装好的R应用程序，出现的console界面和图1-2差不多。

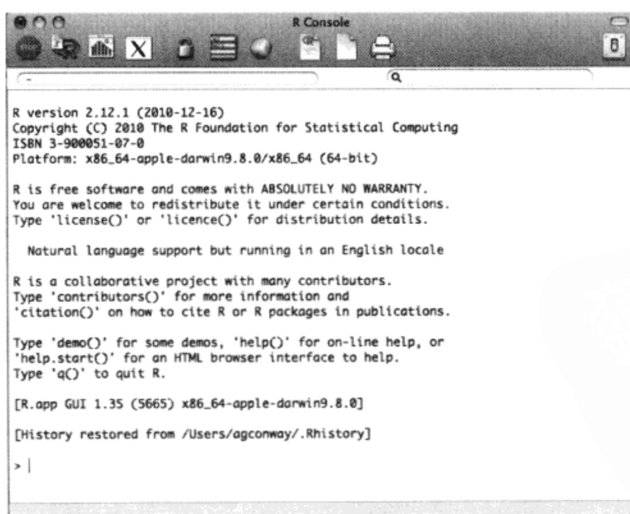


图1-2：Mac OS X系统下64位版本的R控制台界面

注意： 如果你的Mac OS X是自定义安装的，或者你想根据自己的配置自定义安装R，那么建议使用源代码安装。在Mac OS X系统下用源代码安装R同时需要C和Fortran两种语言的编译器，而该操作系统的标准安装中没有这两个编译器。你可以使用Mac Os X开发者工具盘（DVD）来安装这两个编译器，这个盘在Mac Os X原始安装盘中。你也可以从镜像站点上的tools目录下找到必需的编译器然后进行安装。

具备源代码安装所需的编译器之后，其安装过程就与大多数使用命令行安装的软件一样，经典的三个步骤：configure、make和install。用*Terminal.app*进入放源代码的路径，执行以下命令：

```
$ ./configure
$ make
$ make install
```

由于操作系统中用户权限的不同，在执行第一步之前，你可能需要输入系统密码来激活sudo权限。如果在二进制安装和源代码安装过程中遇到了任何问题，都可以到镜像站点的*R for Mac OS X FAQ*页面查询原因。

Linux

许多版本的linux和MAC OS X一样，都预装了R。只需在命令行敲入“R”，就可以加载R控制台。接着，你就可以开始编程了！CRAN上也有Debian、RedHat、SUSE以及Ubuntu这些不同linux版本相应的R安装文件及安装说明。如果你使用其中一个版本进行安装，我们建议参考针对你的操作系统的安装说明，因为不同版本的Linux操作系统，其最佳实践存在的差异相当大。

集成开发环境和文本编辑器

由于R是一种脚本语言，因此在接下来的案例研究中大部分的工作都要用集成开发环境（IDE）或者文本编辑器来完成，而不是直接在R控制台上输入程序。从下一节的内容里，你会发现有些工作适合直接在控制台上完成，比如安装程序包，但是大多数时候你还是愿意在IDE或者文本编辑器中写程序。

在Windows或者Mac OS X环境里运行的R图形用户界面，程序中都有一个基本的文本编辑器。如果你想新建一个空白文档，你可以在菜单上依次选择File→New Document（文件→新建程序脚本），也可以直接单击窗口顶部的空白文档图标（图1-3中高亮处）。作为一个黑客，你应该已经有一个IDE或者文本编辑器了，我们建议你在本书的案例研究过程中从这两者中选择最熟悉的环境来开发。界面上还有很多选项，在此不再一一列举，我们也不想卷入Emacs和Vim之争。

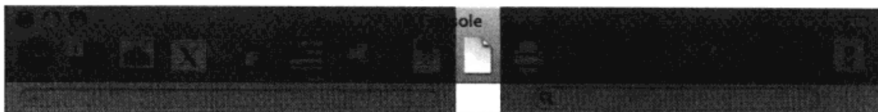


图1-3：R图形用户界面上的文本编辑器图标

安装和加载R程序包

目前已有很多精心设计、维护良好且广泛支持的与机器学习相关的R程序包。在我们要进行的案例研究中，涉及的程序包主要用于：处理空间数据、进行文本分析、分析网络拓扑等，还有些程序包用于与网络API进行交互，当然还有其他很多功能，不胜枚举。因此，我们的任务很大程度上会依赖内置在这些程序包的函数功能。

加载R程序包很简单。实现加载的两个函数是：library和require。两者之间存在细微差别，在本书中，主要差别是：后者会返回一个布尔值（TRUE或FALSE）来表示是否加载成功。例如，在第6章中，我们会用到tm程序包来分词。要加载该程序包，我们既可以用library也可以用require。在下面所举例子中，我们用library来加载tm包，用require来加载XML包，再用print函数来显示require函数的返回值。可以看到，返回的布尔值是“TRUE”，可见XML包加载成功了。

```
library(tm)
print(require(XML))
#[1] TRUE
```

假如XML包还未安装成功，即require函数返回值为“FALSE”，那么我们在调用之前仍需先安装成功这个包。

注意： 如果你刚安装成功R环境，那么你还需要安装较多的程序包才能完成本书的所有案例研究。

在R环境中安装程序包有两种方法：可以用图形用户界面进行安装，也可以用R控制台中的install.packages函数来安装。考虑到本书目标读者的水平，我们在本书的案例研究中会全部采用R控制台进行交互，但还是有必要介绍一下怎么用图形用户界面安装程序包。在R应用程序的菜单栏上，找到Packages & Data→Package Installer（程序包→安装程序包），点击之后弹出如图1-4所示的窗口。从程序包资源库的下拉列表中选择CRAN（binaries）（CRAN（二进制））或者CRAN（sources）（CRAN（源代码）），点击Get List（获取列表）按钮，加载所有可安装的程序包，最新的程序包版本可以从CRAN（sources）（CRAN（源代码））资源库中获取。如果你的计算机上已经安装了所需的

编译器，我们推荐用源代码安装。接着，选择要安装的包，然后点击Install Selected（安装所选包），即可安装。

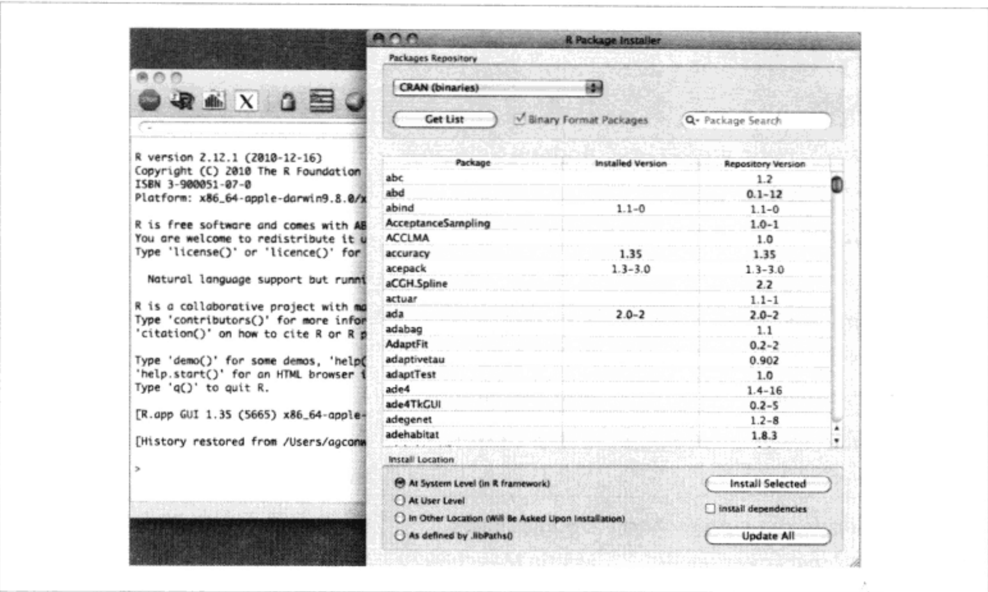


图1-4：用图形用户界面安装R程序包

相比而言，用install.packages函数来安装是一种更佳的方法，因为它在安装方式和安装路径上更为灵活。这种方法的主要优势之一就是既可以用本地的源代码，也可以用CRAN上的源代码来安装。虽然以下这种情况不太常见，但仍然有可能会需要。有时你可能要安装一些CRAN上还未发布的程序包，比如你要将程序包更新到测试版本，那么你必须用源代码进行安装：

```
install.packages("tm", dependencies=TRUE)
setwd("~/Downloads/")
install.packages("RCurl_1.5-0.tar.gz", repos=NULL, type="source")
```

第一行代码中，我们用默认参数从CRAN上安装了tm程序包。tm程序包用于文本挖掘，在第3章将用它来对电子邮件文本进行分类。install.packages中一个很有用的参数是suggests，这个参数默认值是FALSE，如果设置为TRUE，就会在安装过程中通知install.packages函数下载并安装初始安装过程所依赖的程序包。为了得到最佳实践，我们推荐将此参数值一直设置为TRUE，当R应用程序上没有任何程序包的情况下更要如此。

同样还有另一种安装方法，那就是直接使用源代码的压缩文件进行安装。在上一个例子中，我们用作者网站上的源代码安装了RCurl程序包。用setwd函数确保R的工作路径已

设置为保存源代码的目录，然后就可以简单地执行前面的命令从源代码安装了。注意，这里需要改动两个参数。首先，我们必须设置`repos=NULL`来告诉函数不要使用CRAN中任意一个资源库，然后要设置`type="source"`来告诉函数使用源代码安装。

表1-2：本书中用到的程序包

名称	网址	作者	简介及用法
arm	*http://cran.r-project.org/web/packages/arm/*	Andrew Gelman等	用于构建多水平/层次回归模型的程序包
ggplot2	*http://had.co.nz/ggplot2/*	Hadley Wickham	是图语法在R中的实现，是创建高质量图形的首选程序包
glmnet	*http://cran.r-project.org/web/packages/glmnet/index.html*	Jerome Friedman、Trevor Hastie和Rob Tibshirani	包含Lasso和elastic-net的正则化广义线性模型
igraph	*http://igraph.sourceforge.net/*	Gabor Csardi	简单的图及网络分析程序，用于模拟社交网络
lme4	*http://cran.r-project.org/web/packages/lme4/*	Douglas Bates、Martin Maechler和Ben Bolker	提供函数用于创建线性及广义混合效应模型
lubridate	*https://github.com/hadley/lubridate*	Hadley Wickham	提供方便的函数，使在R环境中处理日期更为容易
RCurl	*http://www.omegahat.org/RCurl/*	Duncan Temple Lang	提供了一个与libcurl库中HTTP协议交互的R接口，用于从网络中导入原始数据
reshape	*http://had.co.nz/plyr/*	Hadley Wickham	提供一系列工具用于在R中处理、聚合以及管理数据
RJSONIO	*http://www.omegahat.org/RJSONIO/*	Duncan Temple Lang	提供读写JSON（JavaScript Object Notation）数据的函数，用于解析来自网络API的数据
tm	*http://www.spatstat.org/spatstat/*	Ingo Feinerer	提供一系列文本挖掘函数，用于处理非结构化文本数据
XML	*http://www.omegahat.org/RSXML/*	Duncan Temple Lang	用于解析XML及HTML文件，以便从网络中提取结构化数据

前文已经提到过，在本书中我们会使用一些程序包。表1-2列出了本书的案例研究所用到的所有程序包，包括对其用途的简单介绍，以及查看每个包详细信息的链接。安装所需程序包的数量不少，为了加快安装过程，我们创建了一个简短的脚本来检查每个必需的

程序包是否已安装，若没有安装，它会通过CRAN进行安装。要运行该脚本，先用setwd函数将工作目录设置为本章代码所在的文件夹，再执行source命令，如下所示：

```
source("package_installer.R")
```

如果你还没有安装过程序包，系统可能要求你选择一个CRAN的库。一旦设置完成，脚本就开始运行，你就可以看到所有需要安装的程序包的安装进度。现在，我们就要用R开始机器学习之旅了！在我们开始案例分析之前，我们仍需要回顾一些常用的R相关的函数与操作。

机器学习中的R基础

在本书开篇，我们就提出，学习新技术的最好方式是从你渴望解决的问题或你渴望解答的疑问入手。对任务的较高层次愿景满怀激情，这会让你从案例中有效地学到知识。这里我们要介绍的R基础知识并不会涉及机器学习问题，但是我们将遇到一些R中数据处理和管理相关的问题。在案例研究中可以发现，常常要花很多时间来规整数据，使其格式统一、组织合理以便分析。而花在编写代码、运行分析的时间常常较少。

在接下来这个案例中，我们提出的问题纯属为了好玩。最近，数据服务商*Infochimps.com*发布了一个数据集，其中含有60 000多条不明飞行物（UFO）的目击记录和报道。这份数据时间跨度几百年，地域覆盖全世界。虽然数据涵盖全球，但是主要的目击记录都发生在美国。面对这份数据的时空维度，我们可能会有以下疑问：UFO的出现是否有周期性规律？美国的不同州出现的UFO记录如果有区别，有哪些区别？

我们要探索的是一个很棒的数据集，因为它数量巨大、高度结构化，同时也很有趣。因为它是个大文本文件，而且我们在本书中要处理的数据都属于这一类型，所以练习一下很有用。在这样的文本文件中，常常有一些杂乱的记录，我们会用R中的基本函数和扩展库来清洗和组织原始数据。本节我们会带你一步一步地了解整个简单分析的过程，并试图回答前文提到的两个问题。在本章的*code*文件夹下有一个文件：*ufo_sightings.R*，这是本章所用的脚本源代码。我们首先从加载数据和所需的库开始。

加载程序包和数据

首先，加载的是ggplot2程序包，我们会在最后一步——可视化分析的时候用到它。

```
library(ggplot2)
```

加载ggplot2的过程中，你会注意到这个包还加载了另外两个所需的程序包：plyr和reshape。这些包在R中用于处理和组织数据，同时我们也会在本例中用plyr程序包来完成数据的聚合和组织。

其次，从文本文件*ufo_awesome.tsv*加载数据，文件在本章目录下的*data/ufo/*中。注意一点，该文件字段是制表符分隔的（因此扩展名是*.tsv*），这意味着我们要用read.delim函数来加载数据。因为R直接使用默认值的地方很多，因此我们在脚本中使用函数时要特别注意默认参数的设置。假设我们此前从未用过read.delim函数，为了更好地了解R中参数的意义，则需要阅读帮助文档。或者，假设我们不知道read.delim这个函数的存在，但需要用一个函数将分隔字段的数据读到一个数据框中。R提供了一些有用的函数来解决这些问题：

```
?read.delim                     # 读取一个函数的帮助文档
??base::delim                   # 在所有base包的帮助文档中搜索'delim'
help.search("delimited")        # 在所有的帮助文档中搜索"delimited"
RSiteSearch("parsing text")     # 在R官方网站上搜索"parsing text"
```

在第一个例子中，我们在函数名前面加了个问号。这样就会打开这个函数的帮助文档，这是R中很有用的一个快捷方式。我们也可以在一个程序包中搜索一个特定词，这需要??和::结合使用。双问号表示搜索一个指定的词。在这个例子中，我们用双冒号表示在所有base包的函数中搜索指定词delim。R也允许你进行半结构化的帮助搜索，这会用到help.search和RSiteSearch。help.search能在所有已安装的程序包中搜索指定的词，如上述例子中的delimited。另外，还可以用RSiteSearch搜索R网站上的帮助文档和邮件列表。请注意，千万别以为我们已经把这章要用的函数或者所有R相关的知识都回顾完了。我们极力推荐你自己独立地用这些搜索函数去研究R的其他基本函数。

回到UFO的数据处理上，为了正确地读取数据，我们得给read.delim函数手动设置一些参数。首先，需要告诉函数数据字段是怎么分隔的。我们知道这个文件是制表符分隔文件，所以把sep设置为制表符。然后，read.delim函数在读取数据的过程中会用一些启发式规则把每一行转换成R的数据类型。在本例中，每一行的数据类型都是strings（字符串），但是所有read.*函数都默认把字符串转换为factor类型，这个类型是用来表示分类变量（categorical variable）的，并不是我们想要的。因此，我们需要设置stringsAsFactors=FALSE来防止其转换。实际上，把这个默认值设置为FALSE一般都不会错，尤其是处理不熟悉的数据时。此外，这份数据第一行并没有表头，因此还需要把表头的参数设置为FALSE，以防止R默认把第一行当做表头。最后，数据中有许多空元素，我们想把这些空元素设置为R中的特殊值NA，为此，我们显式地定义空字符串为na.string：

```
ufo<-read.delim("data/ufo/ufo_awesome.tsv",
sep="\t", stringsAsFactors=FALSE,header=FALSE, na.strings="")
```

注意： 上文提到的术语“分类变量”指的是R中的一种数据类型，它表示在一个分类体系中观察到的一个类别成员。在统计学中，分类变量是一个很重要的概念，因为我们会关心：对于一定的类型，它由哪些特定观察值构成。在R中，用factor（因子）类型表示分类变量，其本

质上是给每一个字符串标签赋予一个数值。在本例中，我们用as.factor把某些字符串（比如说州的缩写）转换成分类变量，它会给数据集中每个缩写州名赋予一个独一无二的数值ID。这个过程我们今后会多次碰到。

现在，我们已有一个装着所有UFO数据的数据框了。无论何时，只要你操作数据框，尤其当数据是从外部数据源读入时，我们都推荐你手工查看一下数据。关于手工查看数据，两个比较好用的函数是head和tail。这两个函数会分别打印出数据框中的前六条和后六条数据记录：

```
head(ufo)
        V1       V2                    V3   V4      V5   V6
1 19951009 19951009        Iowa City, IA <NA>    <NA>  Man repts. witnessing "flash..
2 19951010 19951011       Milwaukee, WI <NA>  2 min.  Man on Hwy 43 SW of Milwauk..
3 19950101 19950103         Shelton, WA <NA>    <NA>  Telephoned Report:CA woman v..
4 19950510 19950510        Columbia, MO <NA>  2 min.  Man repts. son's bizarre sig..
5 19950611 19950614         Seattle, WA <NA>    <NA>  Anonymous caller repts. sigh..
6 19951025 19951024 Brunswick County, ND <NA>  30 min.  Sheriff's office calls to re..
```

这个数据框最明显的特点是每一列的名字没有实际意义。参考一下这份数据的文档，我们可以赋予每一列更有意义的标签。给数据框每一列赋予有意义的名称很重要。这样一来，不管对自己还是其他人，代码和输出都有更强的可读性。我们会用到names函数，这个函数既能读取列名，又能写入列名。我们要根据数据文档创建一个与数据集的列名相应的字符串向量，然后把它作为names函数唯一的参数传入：

```
names(ufo)<-c("DateOccurred","DateReported","Location",
"ShortDescription","Duration","LongDescription")
```

从head函数的输出以及用于创建列标签的文档可知，数据的前两列是日期。R中的日期是一种特殊的数据类型，这和其他编程语言没什么不同，因此我们想把日期字符串转换为这种真正的日期数据类型。这需要用到as.Date函数，给它输入日期字符串，它会尝试将其转换为Date对象。在这份数据中，字符串的日期格式是不太常见的YYYYMMDD。因此，需要给as.Date指定一个日期格式字符串，这样它才知道该怎么转换。我们从第一列DateOccurred开始转换：

```
ufo$DateOccurred<-as.Date(ufo$DateOccurred, format="%Y%m%d")
Error in strptime(x, format, tz = "GMT") : input string is too long
```

我们遭遇了第一个错误！尽管不知道错在哪，但是这个错误提到了“input string is too long”（输入字符串过长），这说明DateOccurred列中某些数据太长导致日期格式字符串无法匹配。为什么会出现这种情况呢？我们正在处理的是一个很大的文本文件，因此可能有些数据在原始数据集就是畸形的。如果出现了这种情况，这些畸形的数据就不能

被read.delim函数正确地解析，从而就导致了如上所示的错误。因为我们处理的是真实世界的数据，所以必须手工清理数据。

转换日期字符串及处理畸形数据

要解决这个问题首先必须准确定位这些有缺陷的数据，然后再决定怎么处理。还好我们能从这个提示信息中知道错误的原因是数据“过长”。能够被正确解析的字符串肯定都是8个字符，也就是“YYYYMMDD”这种格式的。为找出有问题的数据行，我们只需要找出那些长度在8个字符以上的字符串。作为一项最佳实践，我们首先要查看畸形数据到底是什么样的，以便对错误原因有更深的理解。在这个例子中，我们还是像之前那样用head函数来检查由逻辑运算返回的结果数据。

然后，为了移除错误的数据行，我们需要用到ifelse函数来构建一个布尔值向量，以便标识出哪些是8个字符的字符串（TRUE），哪些不是（FALSE）。ifelse函数是一个典型的逻辑开关，用于布尔测试，而此处用到的是其向量化版本。我们还会在R中遇到很多向量化操作的例子。这种机制在用于处理数据循环迭代时更占优势，因为这通常（但并非总是）比直接对向量进行循环更有效[注1]：

```
head(ufo[which(nchar(ufo$DateOccurred)!=8
 | nchar(ufo$DateReported)!=8),1])
[1] "ler@gnv.ifas.ufl.edu"
[2] "0000"
[3] "Callers report sighting a number of soft white balls of lights headinginan
easterly directing then changing direction to the west beforespeeding off to the
north west."
[4] "0000"
[5] "0000"
[6] "0000"

good.rows<-ifelse(nchar(ufo$DateOccurred)!=8|nchar(ufo$DateReported)!=8,
                  FALSE,TRUE)
length(which(!good.rows))
[1] 371
ufo<-ufo[good.rows,]
```

在这个搜索过程中，我们用到了一些有用的R函数。我们需要知道DateOccurred和DateReported这两列中每一行字符串的长度，所以用到了nchar函数。如果长度不等于8，则返回FALSE。得到了布尔值向量之后，我们想知道数据集中有多少畸形数据。这样就用到了which函数，它返回一个包含FALSE值的向量。接下来，用length函数计算这个向量的长度就可以知道畸形数据的条数。仅仅371条数据不规范，最简单的办法就是

注1：在R咨询中心有对向量化操作原理的简要介绍：能不能不要循环或者说让循环更快点？（How can I avoid this loop or make it faster?）[F08]

移除并忽略它们。一开始我们可能会担心丢弃这371条畸形数据会不会带来什么后果，但是总共有60 000多条数据，因此简单地忽略它们然后继续转换其他数据，不会有什么影响：

```
ufo$DateOccurred<-as.Date(ufo$DateOccurred, format="%Y%m%d")
ufo$DateReported<-as.Date(ufo$DateReported, format="%Y%m%d")
```

接下来，我们需要清洗、组织目击地点数据。回忆一下我们之前用head函数查看到的美国UFO目击数据，地点数据的形式是“City，State”（城市，州）。我们可以通过R中集成的正则表达式，将字符串拆分成两列，并识别出不规范的数据行。其中识别出不规范数据尤为重要，因为我们只对在美国的UFO目击数据变化趋势有兴趣，而且用这一步的信息也能把美国的数据单独挑出来。

组织目击地点数据

为了继续用上述方式处理数据，首先要定义一个输入为字符串的函数，然后执行数据清洗工作。接下来用apply函数的向量化版本将这个函数应用到地点数据列上：

```
get.location<-function(l) {
   split.location<-tryCatch(strsplit(l,",")[[1]],error= function(e) return(c(NA, NA)))
   clean.location<-gsub("^ ","",split.location)
   if (length(clean.location)>2) {
      return(c(NA,NA))
   }
   else {
      return(clean.location)
   }
}
```

上述函数中有一些细节需要注意。首先，strsplit函数被R的异常处理函数tryCatch所包围。其次，并非所有地点都是“City, State”（城市,州）这种格式，甚至有的数据连逗号也没有。strsplit函数在遇到不符合格式的数据会抛出一个异常，因此我们需要捕获（catch）这个异常。在本例中，对于不包含逗号的数据，我们会返回一个NA向量来表明这条数据无效。接下来，由于原始数据的开头有一个空格，因此用gsub函数（R的正则表达式相关函数之一）移除每个字符串开头的空格。最后，增加一步检查，确保每个返回向量的长度是2。许多非美国地名中会有多个逗号，导致strsplit函数返回的向量长度大于2。在这种情形下，我们依然返回NA向量。

有了定义好的函数之后，我们要用到lapply函数了，它是“list-apply”（应用后返回list链表）的简写，用来对Location列的每一条字符串循环迭代地应用我们定义的函数。前文提到过，R中的apply函数家族相当有用。这些函数的形式都是apply(vector, function)，它们在逐一应用到向量元素上之后，返回特定形式的结果。在本例中，我们用到的是lapply，它返回一个list链表：

```
city.state<-lapply(ufo$Location, get.location)
head(city.state)
[[1]]
[1] "Iowa City" "IA"

[[2]]
[1] "Milwaukee" "WI"

[[3]]
[1] "Shelton" "WA"

[[4]]
[1] "Columbia" "MO"

[[5]]
[1] "Seattle" "WA"

[[6]]
[1] "Brunswick County" "ND"
```

从例子中可以看出，R中的list是一个“键－值”对形式的数据结构。键由双方括号索引，值则包含在单方括号中。在本例中，键就是简单的整数，但是lists也允许用字符串作为键[注2]。虽然把数据存储在list中比较方便，但是却并不是我们想要的，因为我们想把城市和州的信息作为不同的两列加入到数据框中。为此，需要把这个较长的list转换成一个两列的矩阵（matrix），其中city（城市）数据作为其首列：

```
location.matrix<-do.call(rbind, city.state)
ufo<-transform(ufo,USCity=location.matrix[,1], USState=tolower(location.matrix[,2]),
     stringsAsFactors=FALSE)
```

为从list构造一个矩阵，我们用到了do.call函数。和apply函数类似，do.call函数是在一个list上执行一个函数调用。我们还会经常把lapply和do.call函数结合起来用于处理数据。在上述例子中，传入的函数是rbind，这个函数会把city.state链表中的所有向量一行一行地合并起来，从而创建一个矩阵。要把这个矩阵并入数据框中，还要用到transform函数。分别用location.matrix的第一列和第二列创建两个新列：USCity和USState。最后，由于州名字的缩写形式不一致，有的采用大写形式，有的采用小写形式，因此我们用tolower函数把所有的州名字的缩写都变成小写形式。

处理非美国境内的数据

数据清洗要解决的最后一个问题就是处理那些形式上符合“City, State”，但是实际上并不在美国境内的数据。具体来说，有些UFO目击地点在加拿大，而这些数据同样符合这样的形式。幸好加拿大各省的缩写和美国各州的缩写并不匹配。利用这一点，我们可以

注2：　要深入理解lists，请参考《Data Manipulation with R》一书中的第1章[Sep18]。

构造一个美国各州缩写的向量，让USState列数据来匹配这个向量，把匹配上的保留下来，从而识别出非美国地名：

```
us.states<-c("ak","al","ar","az","ca","co","ct","de","fl","ga","hi","ia","id","il","in",
    "ks","ky","la","ma","md","me","mi","mn","mo","ms","mt","nc","nd","ne","nh","nj","nm",
    "nv","ny","oh","ok","or","pa","ri","sc","sd","tn","tx","ut","va","vt","wa","wi","wv",
    "wy")
ufo$USState<-us.states[match(ufo$USState,us.states)]
ufo$USCity[is.na(ufo$USState)]<-NA
```

为了找到USState列中不匹配美国州名缩写的数据，我们用到了match函数。这个函数有两个参数：第一个参数是待匹配的值，第二个是用于匹配的数据。函数返回值是一个长度与第一个参数相同的向量，向量中的值是其在第二个参数中所匹配的值的索引。如果没有在第二个参数中找到匹配的值，函数默认返回NA。在我们的案例中，我们只关心哪些数据返回值为NA，因为这表明它们没有匹配上美国州名。然后，用is.na函数找出这些不是美国州名的数据，并且将其在USState列中的值重置为NA。最后，我们还要把USCity列中对应位置也设置为NA。

现在原始数据框已经处理到位，可以从中只抽取出我们感兴趣的数据了。准确地说，我们想要的数据只是其中发生在美国的UFO记录子集。通过替换前面几步中不符合条件的数据，我们用subset命令创建一个新数据框，只保留发生在美国的数据记录：

```
ufo.us<-subset(ufo, !is.na(USState))
head(ufo.us)
    DateOccurred DateReported            Location  ShortDescription  Duration
1   1995-10-09    1995-10-09       Iowa City, IA              <NA>      <NA>
2   1995-10-10    1995-10-11       Milwaukee, WI              <NA>    2 min.
3   1995-01-01    1995-01-03         Shelton, WA              <NA>      <NA>
4   1995-05-10    1995-05-10        Columbia, MO              <NA>    2 min.
5   1995-06-11    1995-06-14         Seattle, WA              <NA>      <NA>
6   1995-10-25    1995-10-24 Brunswick County, ND             <NA>   30 min.

    LongDescription                     USCity                    USState
1 Man repts. witnessing "flash...       Iowa City                 ia
2 Man on Hwy 43 SW of Milwauk...        Milwaukee                 wi
3 Telephoned Report:CA woman v...       Shelton                   wa
4 Man repts. son's bizarre sig...       Columbia                  mo
5 Anonymous caller repts. sigh...       Seattle                   wa
6 Sheriff's office calls to re...       Brunswick County          nd
```

聚合并组织数据

到目前为止，我们已经把数据组织成可以开始分析的程度了。上一节专注于规范数据的格式，并筛选出要分析的相关数据。本节将继续探索数据，以期进一步缩小我们需要关注的范围。这份数据有两个基本的维度：空间维度（目击事件发生地点）和时间维度

（目击事件发生时间）。上一节我们关注空间维度，这一节将关注时间维度。首先，我们要对DateOccured这列数据应用summary函数，从而对数据的年代范围有个基本认识：

```
summary(ufo.us$DateOccurred)
Min.         1st Qu.      Median       Mean         3rd Qu.      Max.
"1400-06-30" "1999-09-06" "2004-01-10" "2001-02-13" "2007-07-26" "2010-08-30"
```

令人称奇的是，这份数据时间跨度很长，最古老的UFO目击记录可追溯到1400年前！看到这个异常的年份数据，我们不禁要问的是：这份数据在时间上到底是如何分布的？是否值得分析整个时间序列？进行直观判断最快的方法就是构建一个直方图。我们将在第2章详细讨论直方图，不过现在你需要知道的是：直方图就是让你在某个维度上把数据划分成不同的区间，然后观察数据落入每个区间的频度。此处我们感兴趣的维度是时间，因此构建一个直方图，把数据按照时间进行区间划分：

```
quick.hist<-ggplot(ufo.us, aes(x=DateOccurred))+geom_histogram()+
scale_x_date(major="50 years")
ggsave(plot=quick.hist, filename="../images/quick_hist.png", height=6, width=8)
stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```

这里要注意几点。这是我们第一次用到ggplot2程序包，并将在本书所有需要可视化数据的地方用到。在这个例子中，我们构建了一个非常简单的直方图，它只用了一行代码。首先，用UFO数据框作为初始化参数创建一个ggplot对象。接下来，为了美观，把x轴设定为DateOccurred列，因为这一列的频数才是我们所感兴趣的。ggplot2操作对象必须是数据框，而且创建ggplot2对象的第一个参数也必须是一个数据框。ggplot2是对Leland Wilkinson的图语法（Grammar of Graphics）[Wil05]的一种R实现。这说明ggplot2程序包与这种特别的数据可视化哲学是一脉相承的，所有的可视化都是通过一系列的层构建而成的。在图1-5所示的直方图中，初始层便是x轴的数据，即UFO的目击日期。接着用geom_histogram函数添加一个直方图层。在这个例子中，调用geom_histogram函数时使用了其默认参数，而我们在今后会意识到，这样并不是好的选择。最后，由于数据的时间跨度太大，我们用scale_x_date函数将x轴标签的时间周期改为50年。

ggplot对象创建完毕之后，用ggsave函数可以把可视化结果输出到文件里。也可以用print(quick.hist)把可视化结果输出到屏幕上。请注意，在输出可视化结果时会出现警告信息。在直方图中给数据划分区间的方式有很多种，在第2章中我们会详细讨论，而这里输出的警告信息清楚地显示了ggplot2在默认情形下是如何给数据划分区间的。

我们现在就用这份可视化结果对数据一探究竟。

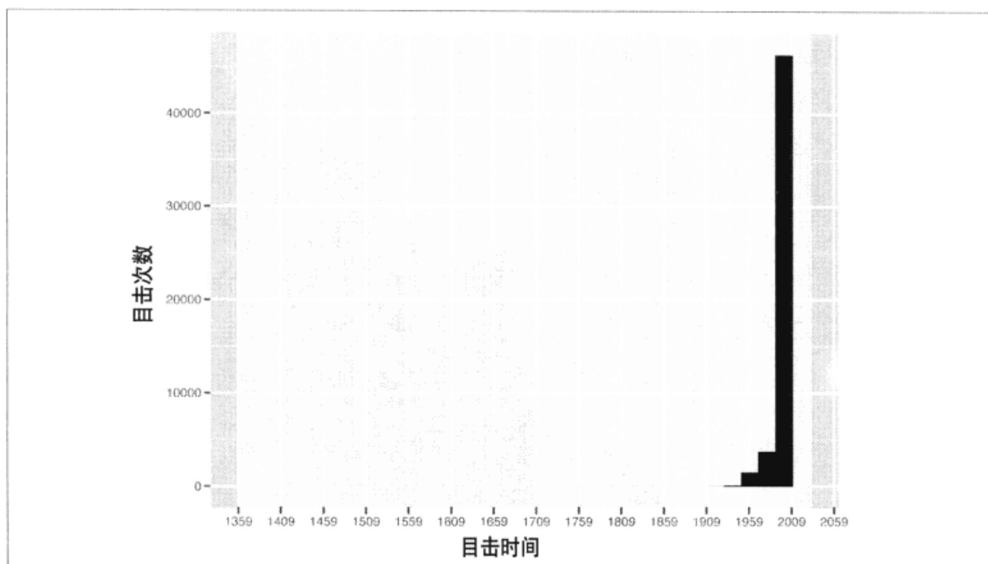


图1-5：UFO数据随时间变化的直方图

从图1-5可以看出来，结论显而易见。绝大部分的目击事件发生在1960~2010年，而其中最主要的又发生在过去20年间。考虑到我们的目的，只需要关注1990～2010年的数据即可。这样我们就能够在分析过程中排除异常值，只比较相近时间区间的数据。和之前一样，用subset函数把符合条件的数据挑出来以构建一个新的数据框：

```
ufo.us<-subset(ufo.us, DateOccurred>=as.Date("1990-01-01"))
nrow(ufo.us)
#[1] 46347
```

虽然这次移除的数据比我们之前清洗数据时丢弃的还要多，但是留下来可供分析使用的样本还有46 000多个。为了看清差异，我们重新对这个子集生成直方图，如图1-6所示。我们注意到这次样本差异更加显著。然后，我们开始组织数据，希望能回答我们的中心问题：美国境内UFO目击记录如果存在周期性规律，会是什么规律？为解答这个问题，我们首先要问的是：所谓“周期性”是什么？有很多方式可以把时间序列按照一定周期聚合：按周、按月、按季度、按年，等等。那么这里用哪种方式来聚合数据最合适呢？DateOccurred列的数据是精确到天的，但是整个数据集的覆盖面上又存在大量的不一致。我们所需的聚合方式应该能让各个州的数据量分布相对均匀。此例中，按year-month（年－月）的聚合方式是最佳选择。这种聚合方式也最能回答我们所提问题的核心，因为按年－月聚合比较容易看出周期性变化。

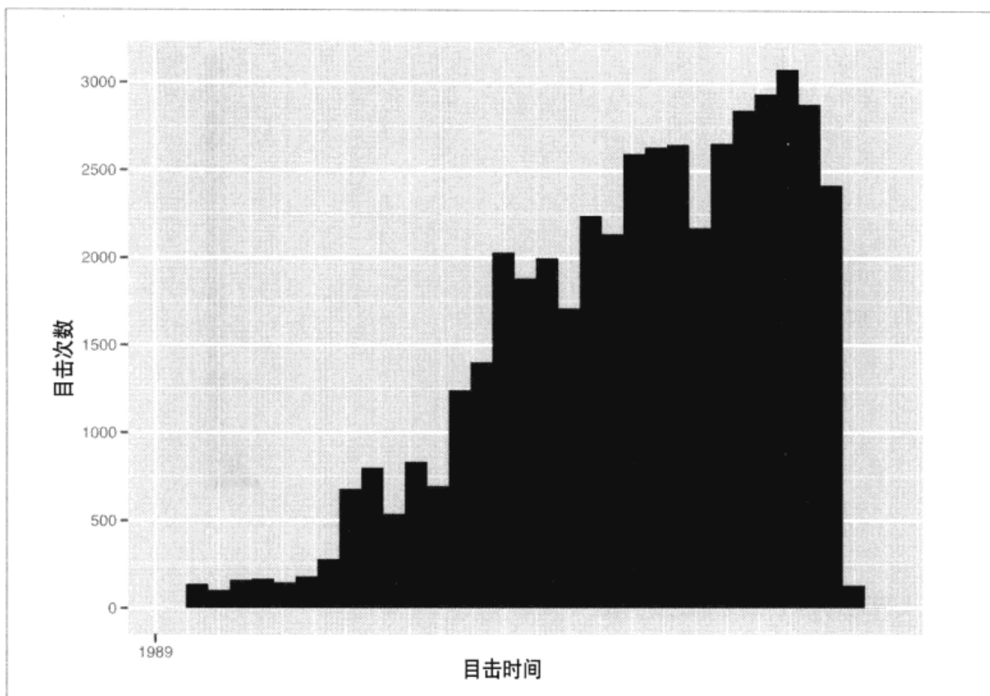


图1-6：UFO子集数据随时间变化的直方图（1990～2010）

我们需要统计1990～2010年每个州每年－月的UFO目击次数。首先，我们要在数据中新建一个列，这个列用于保存和当前数据中一样的年和月。用strftime函数把日期对象转换成一个“YYYY-MM”格式的字符串。我们照旧要设置一个format参数来完成转换：

```
ufo.us$YearMonth<-strftime(ufo.us$DateOccurred, format="%Y-%m")
```

需要注意的是，我们没有用transform函数给数据框添加一个新列。相反，我们简单地引用了一个并不存在的列名，R就自动增加了这个列。这两种给数据框添加新列的方法都有效，而我们会根据具体情况选择使用其中一种方法。接下来我们需要统计每个州在每个年－月期间目击UFO的次数。这里首次用到了ddply函数，此函数是plyr库中的一员，而plyr库在处理数据方面非常有用。

plyr函数家族的作用有点像前几年流行起来的map-reduce风格的数据聚合工具。它们把数据按照一定方式分成不同的组，划分方式对每一条数据都是有意义的，然后对每个组进行计算并返回结果。在本案例中，我们要做的是用“州名缩写”和“年－月”这一新增加的列来给数据分组。数据按照这种方式分好组之后，可以对每个组进行数据统计并把统计结果以一个新列的形式返回。在这里，我们只是简单地用nrow函数按照行数来化简（reduce）每组数据：

```
sightings.counts<-ddply(ufo.us,.(USState,YearMonth), nrow)
head(sightings.counts)
USState     YearMonth     V1
1   ak      1990-01       1
2   ak      1990-03       1
3   ak      1990-05       1
4   ak      1993-11       1
5   ak      1994-11       1
6   ak      1995-01       1
```

现在，我们得到了每个州在每个“年－月”里的UFO目击次数。不过从调用head函数的结果来看，如果直接使用这份统计数据可能存在一些问题，因为其中有大量的值缺失了。比如，我们看到在阿拉斯加州（Alaska），1990年的1月、3月、5月各有一次目击记录，但是没有2月和4月的记录。估计在这期间并没有UFO的目击记录，但是在数据集中的这些位置处也没有包含无UFO目击事件发生的记录，因此，我们得把这些时间记录加上，目击次数就是0。

我们需要一个覆盖整个数据集的“年－月”向量。用这个向量可以检查哪些“年－月”已经存在于数据集中，如果不存在就补上，并把次数设置为0。为此我们要用seq.Date函数创建一个日期序列，然后把格式设定为数据框中的日期格式：

```
date.range<-seq.Date(from=as.Date(min(ufo.us$DateOccurred)),
                     to=as.Date(max(ufo.us$DateOccurred)), by="month")
date.strings<-strftime(date.range, "%Y-%m")
```

有了date.strings这个新的向量，我们还需要新建一个包含所有年－月和州的数据框，然后用这个数据框去匹配UFO目击数据。我们依旧先用lapply函数创建列，再用do.call函数将其转换成矩阵并最终转换成数据框：

```
states.dates<-lapply(us.states,function(s) cbind(s,date.strings))
states.dates<-data.frame(do.call(rbind,states.dates), stringsAsFactors=FALSE)
head(states.dates)
s  date.strings
1  ak         1990-01
2  ak         1990-02
3  ak         1990-03
4  ak         1990-04
5  ak         1990-05
6 ak          1990-06
```

现在，数据框states.dates包含了每一年、每个月和每个州所有组合所对应的所有目击记录。请注意，现在已经有了阿拉斯加州（Alaska）在1990年2月和3月的记录了。要给UFO目击记录数据中的缺失值补上0，我们还需要把这个新数据框和原来的数据框合并。为此，需要用到merge函数，给这个函数输入两个有序数据框，然后它将数据框中相同的列合并。在我们的案例中，两个数据框分别是按照州名的字母顺序以及年－月的

时间顺序排序的。还要告诉函数将数据框中的哪些列进行合并，这就得给参数by.x和参数by.y指定每个数据框中对应的列名。最后，把参数all设置为TRUE，以告诉函数要把没匹配上的数据也包含进来并填充为NA。以下V1列中的值为NA的记录即为没有UFO目击数据的记录：

```
all.sightings<-merge(states.dates,
sightings.counts,
by.x=c("s","date.strings"),
by.y=c("USState","YearMonth"),all=TRUE)
head(all.sightings)
     s        date.strings     V1
1    ak       1990-01          1
2    ak       1990-02          NA
3    ak       1990-03          1
4    ak       1990-04          NA
5    ak       1990-05          1
6    ak       1990-06          NA
```

数据聚合的最后一步只是一些简单的修修补补。首先，我们要把all.sightings数据框的列名改成有意义的名称。方法和本案例一开始的改列名一样。接下来，要把NA值改为0，这里又要用到is.na函数。最后要把YearMonth和State列的数据转换为合适的类型。用前面创建的date.range向量和rep函数创建一个和date.range一模一样的向量，并把年－月字符串转换为Date对象。再强调一次，把日期保存成Date对象要比字符串好，因为前者可以用数学方法进行比较，而后者却不容易办到。同样，州名缩写最好用分类变量表示而不用字符串，因此将其转换为factor类型。下一章会详细讲解factor及其他R数据类型：

```
names(all.sightings)<-c("State","YearMonth","Sightings")
all.sightings$Sightings[is.na(all.sightings$Sightings)]<-0
all.sightings$YearMonth<-as.Date(rep(date.range,length(us.states)))
all.sightings$State<-as.factor(toupper(all.sightings$State))
```

现在，要用可视化方法分析数据了！

分析数据

对于这份数据，我们只用可视化的方式回答前面提出的核心问题。在本书的其余部分，我们会结合数值分析和可视化分析，但由于本案例仅用作介绍R编程的核心范例，因此只需要用可视化分析即可。和之前的直方图可视化不同的是，这里我们要更为深入地使用ggplot2程序包来明明白白地构建一个个的图层。按照这种方式，我们就可以构建这样一个可视化结果：它可以直接回答每个州UFO目击记录随时间变化的周期性规律问题，而且看上去也更加专业。

下面的例子一次性构建好了所有的可视化结果，接下来将分别介绍其中每个图层的意义：

```
state.plot <- ggplot(all.sightings, aes(x = YearMonth,y = Sightings)) +
   geom_line(aes(color = "darkblue")) +
   facet_wrap(~State, nrow = 10, ncol = 5) +
   theme_bw() +
   scale_color_manual(values = c("darkblue" = "darkblue"), legend = FALSE) +
   scale_x_date(breaks = "10 years") +
   xlab("Time") +
   ylab("Number of Sightings") +
   opts(title = "Number of UFO sightings by Month-Year and U.S. State (1990-2010)")
ggsave(plot = state.plot, filename = file.path("images", "ufo_sightings.pdf"),
       width = 14, height = 8.5)
```

依照惯例，第一步还是用一个数据框作为第一个参数来创建一个ggplot对象。这里我们用到了前面所创建的数据框all.sightings。在这里，照样为了绘图的美观，先要创建一个图层，x轴是YearMonth列，y轴是Sightings数据。然后，为了表现各州的周期性变化，我们给每一个州绘制一幅曲线图。这样就方便观察每个州UFO目击数随时间变化的峰值、平缓区、波动区。为此，我们要用到geom_line函数，并将color的值设置为“darkblue”（深蓝色）以便让图形更显眼。

通过整个例子我们看到UFO目击数据相当大，覆盖了相当长一段时期内美国所有州的记录。有鉴于此，我们需要设计一种方法，将可视化结果进行拆分，从而既能观察到每个州的数据，又能比较不同州之间的数据。如果我们把所有数据绘制在一个图板中，那很难观察到差别。为了确认这种方式的确不可行，将上面代码块的前两行[译注1]中的color="darkblue"替换为color=State，并在控制台输入> print(state.plot)，然后运行。更好的方法是把每个州的数据单独绘制图形，并将图形在网格中按顺序放好，这样易于进行比较。

为了创建出一个分块绘制的图形，我们用到facet_wrap函数，并指明图形面板的构造使用了变量State，它是一个factor类型，即分类变量。我们也要明确定义网格的行列数，这个比较容易，因为我们已经知道了一共有50个不同的图形。

ggplot2程序包有很多不同的图形绘制主题。默认的绘制主题就是我们在第一个绘制例子中用到的：灰色背景、深灰色网格线。严格来讲，这只是个人喜好问题，但是在这里我们倾向用白色背景，因为这样可以在可视化结果中更容易看到数据之间的不同之处。我们添加了theme_bw层，这个图层会用白色背景和黑色网格线绘制图形。在操作ggplot2更熟练之后，我们推荐你多尝试不同的绘制主题，然后找到自己最钟爱的一个。

译注1：原书中为第一行，实际上第一行是无法成功运行的。

其余图层的创建只是为了锦上添花，让可视化结果在视觉和感觉上都更专业一些。虽然无人对此苛求，但正是这些细节让绘制的可视化数据更贴近专业性而非业余。scale_color_manual函数用来指明字符串“darkblue”相当于网页安全色“darkblue”。虽然这样看上去有些重复、多余，但这正是ggplot2的设计核心，它需要明确定义诸如颜色这样的细节。事实上，ggplot2倾向于用颜色来区别不同的数据类型或分类，因此它偏向于用factor类型来指明颜色。在前面的案例中，我们明确将颜色定义成了一个字符串类型，因此还要用scale_color_manual函数定义这个字符串的值。

我们同样用scale_x_date函数指明可视化结果中主要的网格线。因为这份数据时间跨度是20年，所以要将其间隔设置为5年，然后，将四位数格式的年份作为坐标系的刻度标签。接着用xlab和ylab函数分别将x轴的名称定为Time（目击时间），y轴的名称设置为Number of Sightings（目击次数）。最后，用opts函数给整幅图赋予一个标题。opts函数还有很多的选项可设置，在后面的章节中我们也会用到其中一些，但是还有更多的功能本书没有涉及。

所有图层都已经完成，现在用ggsave函数把结果渲染成图像并分析数据。

分析之后，可以发现很多有趣的现象（见图1-7）。我们看到加利福尼亚州（California）和华盛顿州（Washington）的UFO目击次数明显多于其他各州。而这两个州之间也有一些有趣的区别，加利福尼亚州UFO目击次数在时间分布上比较随机，但在1995年之后又稳步增长，然而华盛顿州的UFO目击次数随时间的周期性变化则始终如一，自1995年开始，其曲线的波峰和波谷都比较有规律。

我们也观察到，许多州的UFO目击次数都出现过突增情况。例如，亚利桑那州（Arizona）、佛罗里达州（Florida）、伊利诺伊州（Illinois）以及蒙大拿州（Montana）在1997年出现过次数突增，密歇根州（Michigan）、俄亥俄州（Ohio）以及俄勒冈州（Oregon）在1999年末出现过相似的突增情况。但是在这些州中，只有密歇根州（Michigan）和俄亥俄州（Ohio）在地理上邻近。如果我们不相信这些是天外来客造访的结果，那有没有其他可能的解释呢？兴许美国人在世纪之交对天空很警觉，常常仰望天空，因此才有了比以前多的错误目击记录。

然而，如果你赞同外太空访客真的经常造访地球这种说法，的确有证据可以激发你想继续探究的好奇心。实际上，通过对地区聚类，也能得到证据发现在美国许多州都有令人称奇的、稳定的周期性目击记录。这些目击记录似乎真的包含了一些有意义的模式。

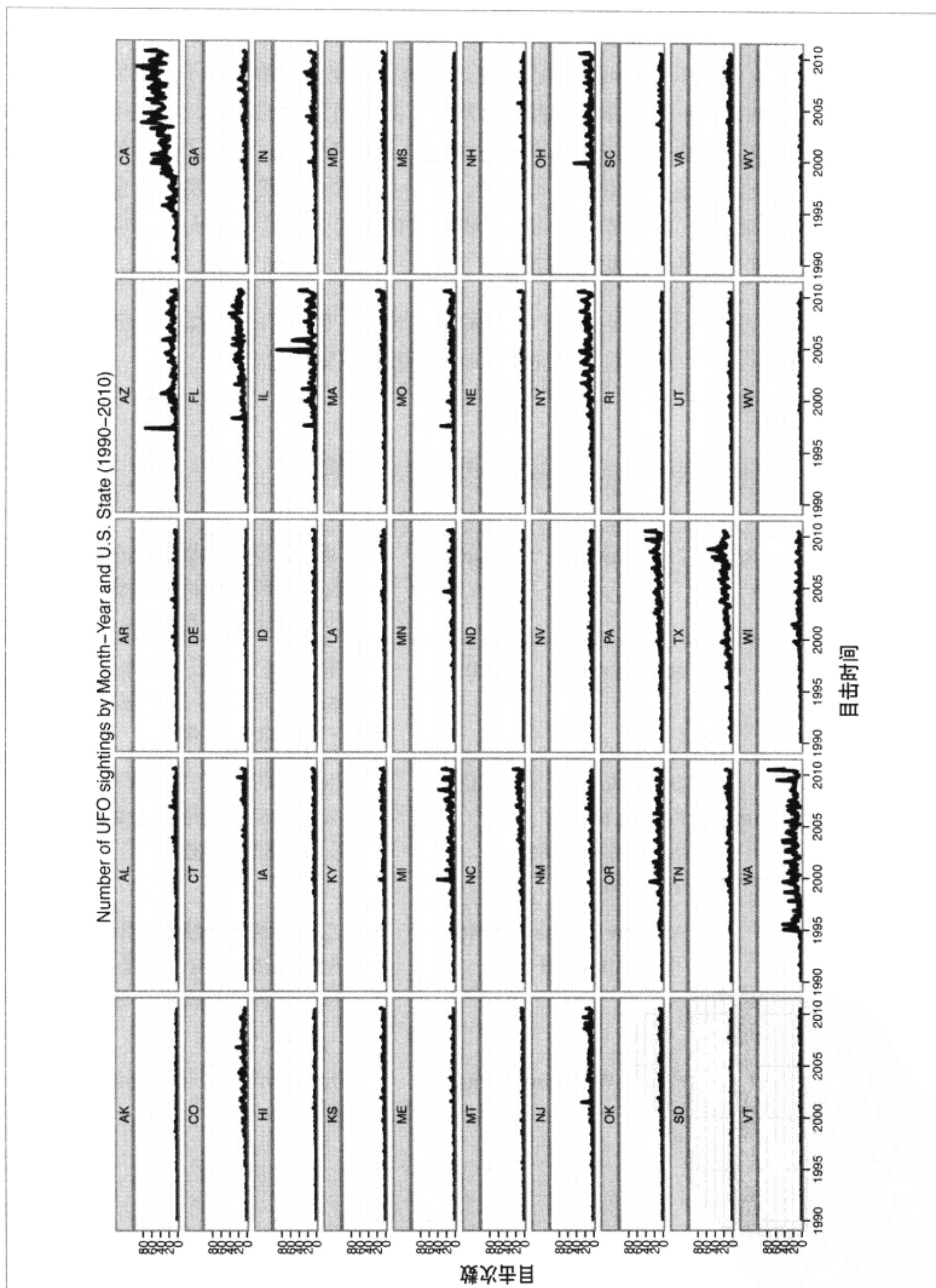


图1-7：美国各州每年－月UFO目击次数（1990~2010年）

深入学习R的参考书目

学习完这个入门案例绝不能说明我们已经把这门语言完全介绍给大家了。我们只是用这个案例介绍了一些R中加载、清洗、组织以及分析数据所用到的模式。在接下来的其他章节里，我们不仅会多次重复用到其中的许多函数、处理过程，我们还会介绍一些在这个案例里还没涉及的知识。在继续学习之前，对于那些希望获得更多实践机会以增强对R熟悉程度的读者而言，有很多很棒的学习资源可供参考。这些资源大体上可以分为参考书、文献以及网上资源，如表1-3所示。

在第2章我们将介绍探索性数据分析。本章的案例包含了较多的数据探索，但是我们处理得比较简单快速。下一章我们会更加慎重地对待数据探索的过程。

表1-3：R参考资源

书名	作者	引用	简介
文献参考资源			
《Data Manipulation with R》	Phil Spector	[Spe08]	对此前涉及的数据处理问题有深入的回顾与探讨，对此前未涉及的一些技术也有介绍
《R in a Nutshell》	Joseph Adler	[Adl10]	详细探讨R中所有基本函数。这本书采用了R手册的形式并加入了很多实际案例
《Introduction to Scientific Programming and Simulation Using R》	Owen Jones、Robert Maillardet和Andrew Robinson	[JMR09]	和其他的R入门文献不同，这本书首先专注于语言本身，然后才设计了模拟实践环节
《Data Analysis Using Regression and Multilevel/Hierarchical Models》	Andrew Gelman和Jennifer Hill	[GH06]	这本书侧重于统计分析，但是所有例子都使用R，它是相当不错的一份学习语言和方法的资源
《ggplot2: Elegant Graphics for Data Analysis》	Hadley Wickham	[Wic09]	用ggplot2可视化数据的专门手册

表1-3：R参考资源（续）

书名	作者	引用	简介
网上参考资源			
《An Introduction to R》	Bill Venables和David Smith	*http://cran.r-project.org/doc/manuals/R- intro.html*	源自R核心团队，是一份对R语言广泛的、时时更新的入门资料
《The R Inferno》	Patrick Burns	*http://lib.stat.cmu.edu/S/Spoetry/Tutor/R_inferno.pdf*	一份很不错的R入门资料，适合资深程序员阅读。其摘要一语中的："如果你正在使用R，而且感觉生不如死，那么这份资料就是来拯救你的。"
《R for Programmers》	Norman Matloff	*http://heather.cs.ucdavis.edu/~matloff/R/RProg.pdf*	和《The R Inferno》比较类似，这份资料也面向其他语言的资深程序员
"The Split-Apply-Combine Strategy for Data Analysis"	Hadley Wickham	*http://www.jstatsoft.org/v40/i01/paper*	很不错的入门资料，来自plyr程序包的作者，在plyr的情景中用很多例子介绍了map-reduce模式
"R Data Analysis Examples"	UCLA Academic Technology Services	*http://www.ats.ucla.edu/stat/r/dae/default.htm*	一份"罗塞塔石碑"式的入门读物，面向的是在诸如SAS、SPSS以及Stata等其他统计语言编程平台上富有经验的人

第2章

数据分析

分析与验证

在数据处理方面，一个屡试不爽的方法就是：把任务清晰地分解成分析和验证两步。把数据处理分为分析和验证由著名的John Tukey[注1]首先提出，他强调要为实际数据分析设计简单的工具。在John Tukey看来，数据分析这一步要做的就是用摘要表（summary table）和基本可视化方法从数据中寻找隐含的模式。本章揭示如何用R中的基本方法来建立数据的摘要表，然后再讲解怎样从这个表中看出门道。之后，将列举R中一些用于可视化数据的工具和方法，同时，还会简要介绍基本可视化模式。

在开始第一次数据集分析之旅前，我们得提醒，一个真实存在的危险在时时窥伺你：你找到的模式也许并不存在。人类的大脑天生就能发现宇宙中的模式，甚至在不太可能出现模式的地方也能发现。看一眼白云，眨眼间就能发现云的形状，这跟知道多少统计学知识无关。很多人都深信自己可以在普通的字里行间，比如莎翁的戏剧，发现隐藏的信息。因为人类很容易发现一些经不起仔细推敲的模式，所以就不能只有数据分析这一步，必须还要有验证过程。可以这样来看待数据验证：数据分析阶段的可视化结果杂乱——有时甚至是毫无章法，这种情况让我们处理起来有些吃力，需要厘清思路，此时数据验证就会发挥作用。

数据验证通常有两种方法：

注1： 同样也发明了bit（位）这个词。

- 如果你认为自己在一个新的数据集上发现了模式，那就用另一批数据来测试这个模式的正规模型[译注1]。
- 利用概率论来测试你在原始数据集中发现的模式是否只是巧合[译注2]。

相比数据分析，因为数据验证对数学水平要求较高，所以本章只涉及数据的分析方法。也就是说在具体的案例中我们只关注数据的数值摘要（numeric summary）和一些标准的可视化方法。我们讲到的数值摘要就是一些基本的统计项目：均值（mean）和众数（mode）、百分位数（percentile）和中位数（median）、标准差（standard deviation）和方差（variance）。我们要用到的可视化工具也是最基本的，你可能在统计学入门课程上已经学过了：直方图、核密度估计以及散点图。这些简单的可视化方法可能一度得不到重视，但是我们会让你相信，仅用这些基本的方法也能从数据中挖掘出有用的信息。本书后面的章节才会涉及许多较高深的技术，但是，要训练数据分析的直觉，最好就是用简单的方法去操作数据。

什么是数据

在介绍数据分析的基本方法之前，我们需要对“数据”这个概念统一认识。对“数据”这个概念所有可能的定义可以用一整书来论述，因为对任何所谓的“数据集”你都可能有一堆重要的问题要问。例如，你可能经常想知道你手中的数据是如何产生的，你也想知道这些数据能否代表真正要研究的数据总体。通过研究亚马逊印第安人婚史，你可能已经掌握了关于他们社会结构的很多知识，但是能否从中得到一些适用于其他文明的知识却不得而知。对数据进行解释需要对数据来源有一定的了解。通常，唯一能区别因果关系和相关关系的方法就是要知道数据由何而来：是从实验中得到的，还是因没有实验数据而直接观察记录而来的。

虽然这都是些有趣的问题，而且我们也希望你终有一天能够具备这些知识，[注2]但是在本书中我们完全不会涉及数据采集。本章先假定数据分析这个微妙的哲学问题与预测问题完全无关，后者我们会用机器学习技术来解决。出于实用性考虑，我们要在本书余下内容中统一采用以下定义：“数据集”无非是一张充满数字和字符串的大表，表中每一行是现实世界中的单个观测记录，并且每一列是观测记录的一个属性。如果你很熟悉数据库，那么我们对数据的这个定义在直觉上应该符合你所熟悉的数据库表结构。如果你还担心你的数据集可能还不仅是单张表，那么我们先假定你已经用过R中的merge、SQL中的JOIN系列操作或者我们此前提到的其他工具，创建了类似单张表的数据集。

译注1：即交叉验证。

译注2：即假设检验。

注2：　你若感兴趣，我们推荐阅读Juden Pearl的《Causality》（因果）[Pea09]。

我们称这个定义为“数据方块”（data as rectangle）模型。虽然这个观点大大简化了，但是却可以在数据分析时非常形象地帮助我们想出许多好主意，我们希望它可以让许多原本抽象的概念变得具体一些。“数据方块”模型还有另一个作用：它让我们可以自由地徜徉于数据库设计的思维和纯数学思维之间。如果你还在为不熟悉矩阵而发愁，大可不必，本书通篇，你都可以把矩阵看成是一个二维数组，即一张大表。只要我们一直想象自己处理的是一个矩形的数组，我们就能使用很多强大的数学工具而不用关心其中具体的数学运算是如何执行的。例如，尽管我们所要研究的每一项技术都可以看做矩阵乘法问题，无论是标准线性回归模型还是现代矩阵分解技术，其中现代矩阵分解技术最近因Netflix奖而声名鹊起，但是只在第9章才简单地介绍了矩阵的乘法。

因为我们把数据当做方块、表、矩阵，所以会交替使用这几个词，恳请读者朋友耐心些。当我们谈起数据时无论用的是哪个词，都请你牢记，它都表示类似表2-1的形式。

表2-1：本书作者

姓名	年龄
Drew Conway	28
John Myles White	29

由于数据由矩形组成，因此我们可以轻松地把运算操作通过绘图的方式表示出来。一份数据的数值摘要就是把表中的所有行精简为只有几个数字——数据集的每一列常常就精简为一个数字。图2-1是这类型数值摘要的例子。

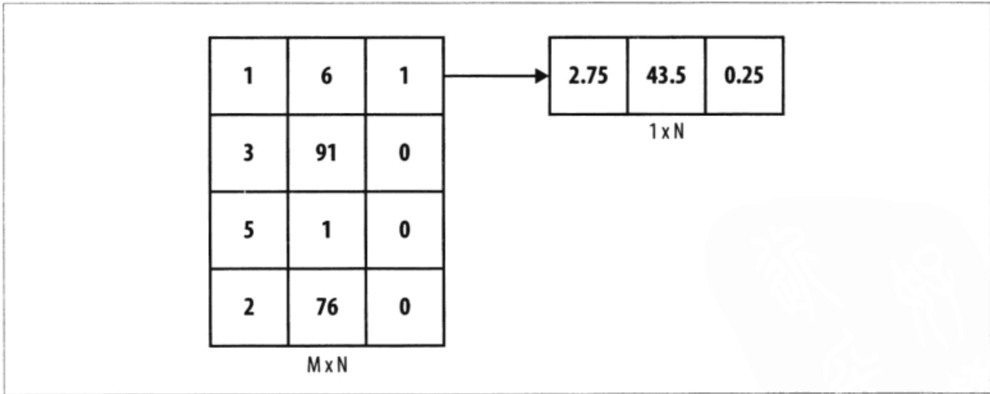


图2-1：将每一列摘要成一个数字

与数值摘要相对的是另一种可视化摘要方法，它把数据集中所有数据的某一列概括成一张图。单列可视化摘要的例子如图2-2所示。

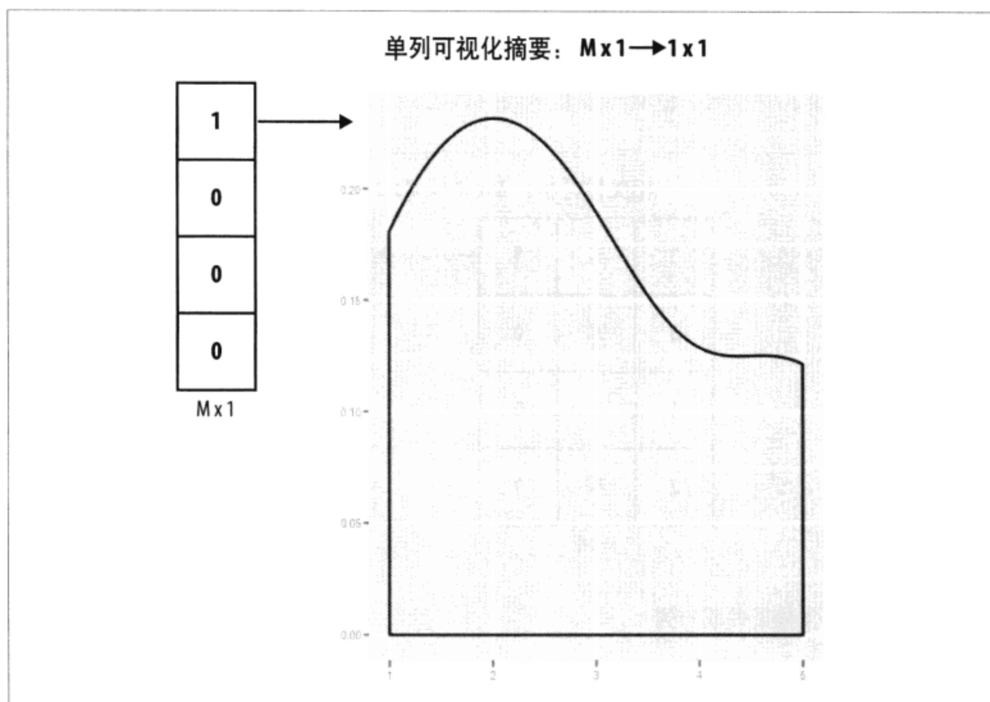


图2-2：将一列摘要成一张图

除了分析单列的方法之外，还有很多方法用于分析数据集中多列之间的关系。例如，计算两列的相关性，可以把表中任意两列转换成一个数字，这个数字表征了这两列之间关系的强度，如图2-3所示。

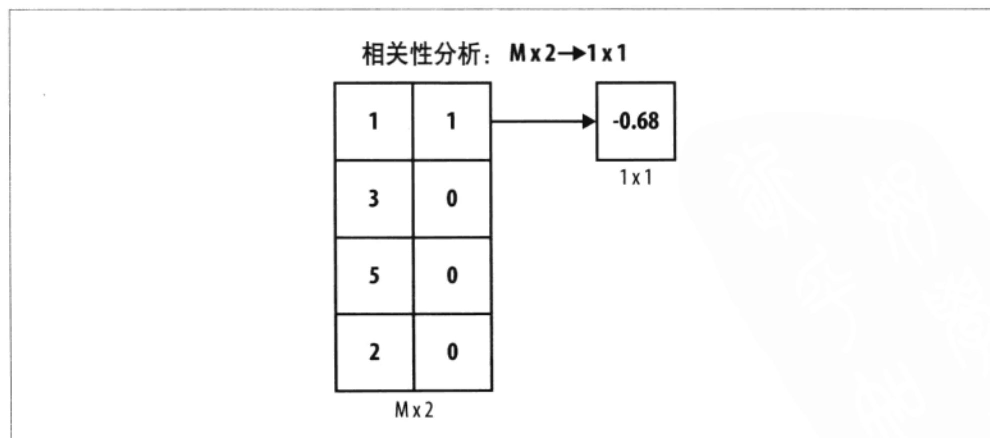


图2-3：相关性：将两列摘要成一个数字

还有其他更为深入的方法。如果你觉得数据中存在很多冗余，那么你可能还需要减少其中的列数。将数据中的很多列变成几列甚至一列的做法称为降维，对此将在第8章详细讲解。图2-4列举了一个例子说明降维要达到的目的。

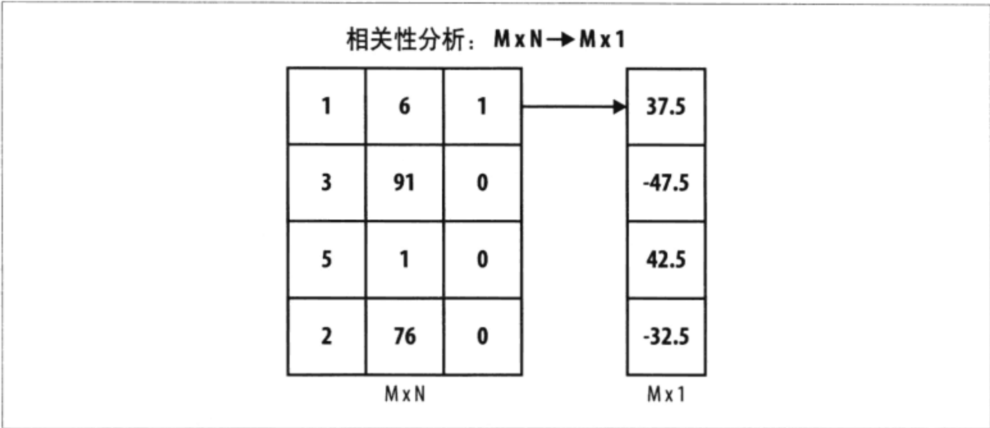


图2-4：降维：将多列摘要成一列

如图2-1~图2-4这四个图所示，摘要统计和降维是截然不同的两个方向：摘要统计要传达的是所有数据在某一列（某个属性）上的特点如何；降维要做的是把数据集中所有列（属性）转换成少数几列，得到的列数据对每一行来说都是唯一的。分析数据时，这两种方法都可能有用，因为它们都可以将海量数据变得一目了然。

推断数据的类型

在你要对新数据集进行下一步操作之前，首先要弄清楚这个表中的每一列所代表的含义。有人喜欢把这称作数据字典，就是说你可以在这里给数据集中每一列数据都存放一条简洁易懂的描述信息。例如，表2-2所示的是一个没有标签的数据集：

表2-2：无标签数据

...	...	...
“1”	73.847017017515	241.893563180437
“0”	58.9107320370127	102.088326367840

在没有任何提示信息的情况下，真的很难知道表中的数字表示什么含义。事实上，首先要搞清楚的是每一列的数据类型：第一列似乎仅包含字符0或1，但它真的是字符串吗？在第1章UFO的例子中，我们首先给数据集每一列打上了标签。当我们面对一份没有标

签的数据集时，我们可能会用到R中的一些类型判断函数。三个最重要的类型判断函数如表2-3所示：

表2-3：R的类型判断函数

R函数	简介
is.numeric	输入向量是数字则返回TRUE，数字包括整数和浮点数；否则返回FALSE
is.character	输入向量是字符串则返回TRUE，R中并没有单字符数据类型；否则返回FALSE
is.factor	输入向量是因子（factor）的某个值时返回TRUE，因子在R中用于表示分类信息；如果你用过SQL中的枚举类型，那么可以把因子看成类似的类型。它在内部隐藏的表示方式和语义上都与字符串向量有区别：R中大多数统计函数的操作对象都是数值向量或因子向量，而不是字符串向量。输入不是因子的某个值时返回FALSE

知道每一列的基本数据类型可能对于我们进行下一步操作非常重要，因为单个R函数常常因为输入数据的类型不同而进行不同的操作。在使用某些R内置函数之前，当前数据集中存储的这些0和1字符需要转换成数字，但是当使用另外一些R内置函数时，它们又会转换成因子类型。从某种程度上来说，这种在不同数据类型之间来回转换的做法是机器学习中处理分类时司空见惯的做法。许多真正充当标签或起分类作用的变量都以数学方式编码成数字0和1。可以把这些数字想象成布尔值：0表示正常电子邮件，1表示垃圾电子邮件。这是用0和1对一个对象的定性属性进行描述的一种方法，这种方法在机器学习和统计学中叫做虚拟变量编码（dummy coding）。虚拟编码系统和R中的因子还是有区别的，后者是采用文字标签来表达对象的定性属性。

警告： R中的因子类型可以当做标签，但是这些标签在后台实际上还是编码为数值型：当程序员读取标签时，这些数值自动地映射为一个字符串索引数组中对应的字符串标签。因为于R在后台采用的是数值编码，所以你异想天开地把R因子的标签转换成数值可能会产生奇怪的结果，原因是你得到的数值可能由你自己的编码方案产生，而非原来与R因子的标签关联的数值。

表2-4～表2-6所示的是同样的数据，但是采用了三种不同的编码方案。

表2-4：因子编码

MessageID	IsSpam
1	“yes”
2	“no”

表2-5：虚拟变量编码

MessageID	IsSpam
1	1
2	0

表2-6：物理学家的编码

MessageID	IsSpam
1	1
2	−1

在表2-4中，`IsSpam`就直接当做R中的因子处理，这样是一种表示定性区别的方法。而在实际中它既有可能以因子类型载入，也有可能以字符串类型载入，这取决于你所用的载入函数（详情请参照第1章中对参数`stringsAsFactors`的介绍）。每拿到一份新数据时，你都要先决定怎么用R处理每一列，再决定是以因子类型还是以字符串类型加载其值。

注意： 如果你不确定到底该以何种类型加载时，最好的方法是先以字符串类型加载，之后根据需要再转换成因子类型。

在表2-5中，IsSpam仍然是一个定性概念，但是却以数字形式表示了布尔型的区别：1表示IsSpam是true，而0表示IsSpam是false。实际上，很多机器学习算法都要求定性数据是这种编码方式。例如，R中默认用于逻辑回归和分类算法的函数glm就假定变量是虚拟变量，这些将会在第3章讲到。

最后，表2-6展示了对相同定性概念进行数值编码的另一种方式。在这种编码系统中，人们用+1和−1，而不用1和0。这种对定性概念编码的风格很受物理学家青睐。当你看过的机器学习书籍够多，最终会遇到这种风格。但是本书会完全摒弃这种风格，因为在变量的不同表示方式间来回切换会导致无谓的混淆。

推断数据的含义

尽管你已经搞清楚了每一列的数据类型，但是对其含义可能仍然不甚了解。确定一张无标签的数字表到底在描述什么比大家想象的要困难得多。我们再看看之前提到的表，见表2-7：

表2-7：无标签数据

...	...	...
“1”	73.847017017515	241.893563180437
“0”	58.9107320370127	102.088326367840

假如我们告诉你以下这些信息：a）每一行代表一个人；b）第一列是一个虚拟变量，表示这个人是男性（值为1）还是女性（值为0）；c）第二列是每个人的身高，以英寸为单位；d）第三列是每个人的体重，以磅为单位。知道这些之后，你再来看这张表，是否觉得豁然开朗？把这些数字放在适当的上下文之中，它们瞬间就变得有意义了，而且这也让你对这些数字所描述的对象有了一定的想法。

但是，遗憾的是，有时候你得不到这些解释性信息。遇到这种情况，我们能告诉你的方法就只有一个：用人类的直觉，再辅以大量的Google搜索。不过，在查看了这类数值摘要表或可视化摘要图之后，这两种摘要表中每一列的含义不明确，你的直觉会大幅改善，这也算是因祸得福。

数值摘要表

要弄清楚一份新数据的意义，最好的方法之一就是计算所有列的数值摘要。R很适合完成这个操作。如果数据中只有一个列向量，summary函数会产生出一些值，这些值的意义再明白不过了，你应该首先看一看：

```
data.file <- file.path('data', '01_heights_weights_genders.csv')
heights.weights <- read.csv(data.file, header = TRUE, sep = ',')
heights <- with(heights.weights, Height)
summary(heights)

#Min. 1st Qu.          Median   Mean     3rd Qu. Max.
#54.26 63.51 66.32     66.37    69.17    79.00
```

对一个数值向量调用R中的summary函数之后就会得到上述例子中这些数值结果：

1. 向量中的最小值
2. 第一个四分位数（也称作第25个百分位数，即大于25%的数据中最小的那个）
3. 中位数（也称作第50个百分位数）
4. 均值
5. 第3个四分位数（也称作第75个百分位数）
6. 最大值

如果你想从数据中快速地得到一份数值摘要，这些值基本上就够了。还缺少的值是标准差，本章稍后会定义一份数值摘要表。在接下来的内容里，我们会教大家怎样单独计算summary函数所产生的每一个数值，并告诉大家它们都有什么意义。

均值、中位数、众数

区分均值和中位数是所有统计学入门课程中最乏味的内容之一。的确需要一些时间才能更加熟悉这些概念，但是当你真正处理数据时，我们相信你还是需要将二者区分开来。考虑到更好地让读者学到知识，我们试图通过两种不同的方式深化并强调这两个术语的意义。第一，如何通过算法方式计算均值和中位数。对大多数黑客来说，用代码表达想法比用数学符号更加自然，所以我们认为，自己写函数来计算均值和中位数，这样可能会让你更加透彻地理解这两个统计学概念的定义公式。第二，在本章末尾还会通过数据的直方图和密度图来揭示两者的区别。

计算均值相当简单。在R中，一般用mean函数计算均值。当然，把均值放在一个黑盒子函数里面计算不太能直观地体会到均值的含义，因此我们自己实现mean函数，命名为my.mean。因为相关概念在R中已有其他函数可以计算：sum和length，所以仅需一行R代码。

```
my.mean <- function(x) {
  return(sum(x) / length(x))
}
```

就这一行代码，均值的含义就已明了：只需把向量中的值求和，再除以长度。想必你已经知道，这个函数用于产生向量x中所有数的平均数。均值太容易计算了，因为它和列表中数字的排序毫无关系。

中位数就刚好相反：它完全依赖于列表元素的排序。在R中，一般用median函数计算中位数，但是我们要实现我们自己的版本，称之为my.median：

```
my.median <- function(x) {
  sorted.x <- sort(x)

  if (length(x) %% 2 == 0)
  {
    indices <- c(length(x) / 2, length(x) / 2 + 1)
    return(mean(sorted.x[indices]))
  }
  else
  {
    index <- ceiling(length(x) / 2)
    return(sorted.x[index])
  }
}
```

只看代码行数就知道计算中位数比计算均值要复杂一些。第一步，对向量排序，因为中位数本质上是有序向量中间的那个数。这也是中位数称为第50个百分位数、第二个四分位数的原因。一旦把向量排好序，就可以顺理成章地计算任何一个百分位数或四分位数，只需根据向量长度从某处将列表拆分成两段即可。要计算第25个百分位数（即第一个四分位数），只需把列表按照长度拆分并取前四分之一即可。

以长度为依据的不规范定义存在一个问题，那就是当数据长度是偶数时此定义就行不通。如果没有一个数明显地处于数据集中间，你需要为此专门设计一个计算方法。在上述例子的代码中，我们专门对这种长度为偶数的情况进行了处理：若列表长度为偶数，则在数据集中间有两个数——这两个数所在位置相当于列表长度为奇数的情况下的中位数——对这两个数求均值即是中位数。

为了解释得更为清楚，下面举一些简单的例子，第一个例子的中位数是中间两个数的均值，另一个例子的中位数就是中间那个数：

```
my.vector <- c(0, 100)
my.vector
# [1] 0 100
mean(my.vector)
#[1] 50
median(my.vector)
#[1] 50
my.vector <- c(0, 0, 100)
mean(my.vector)
#[1] 33.33333
median(my.vector)
#[1] 0
```

回到原来的身高和体重数据集，计算身高的均值和中位数。顺便检验一下代码是否正确：

```
my.mean(heights)
#[1] 66.36756
my.median(heights)
#[1] 66.31807
mean(heights) - my.mean(heights)
#[1] 0
median(heights) - my.median(heights)
#[1] 0
```

在这个案例中，均值和中位数非常接近。我们稍后会简单解释为什么这份数据的均值和中位数本就应该是接近的。

介绍了以上两个重要的统计数值，你可能奇怪为什么还不介绍众数。其中一个原因是：众数不像均值和中位数那样总是可以用我们处理过的那些向量来简单定义。因为它不易自动化计算，所以R中并没有计算数值向量众数的内置函数。

注意： 对任意向量定义众数是比较困难的，因为若要用数值来定义众数就要求向量中的数字重复出现。但是当向量中的数值是任意浮点数时，不太可能任意值都会在向量中重复出现。因此，众数在许多种数据集上仅有可视化定义。

总之，如果你仍不确定众数的数学含义或其理论意义，你就假定它是在数据集中出现次数最多的那个数。

分位数

前面已经说过，中位数就是出现在数据集的50%那个位置的数。为了对数据范围有更深入的认识，你可能想知道数据中最小的那个数是什么。这就是数据集的最小值，这可以用`min`函数计算：

```
min(heights)
#[1] 54.26313
```

而数据集中最高/最大的那个数则可以用`max`计算：

```
max(heights)
#[1] 78.99874
```

两者联合确定了数据的范围：

```
c(min(heights), max(heights))
#[1] 54.26313 78.99874
range(heights)
#[1] 54.26313 78.99874
```

对此可以从另一个角度进行理解：在数据集里，`min`（最小值）指的是0%的数据都小于它的数字，`max`（最大值）指的是100%的数据都比它小的数字。由此可以自然地联想到：如何找出数据集中N%的都小于它的数？答案是使用R中的`quantile`（分位数）函数。第N个分位数就表示数据集中有N%的数据小于它。

默认情况下，`quantile`会告诉你数据集的0%、25%、50%、75%以及100%位置处的数据：

```
quantile(heights)
#       0%       25%       50%       75%      100%
#54.26313  63.50562  66.31807  69.17426  78.99874
```

要得到其他位置的分位数，需要给`quantile`的另一个参数`probs`传入截取位置：

```
quantile(heights, probs = seq(0, 1, by = 0.20))
#       0%       20%       40%       60%       80%      100%
#54.26313  62.85901  65.19422  67.43537  69.81162  78.99874
```

这里用seq函数在0～1之间产生了一个步长为0.2的序列：

```
seq(0, 1, by = 0.20)
#[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

分位数虽然在统计学传统教程中不如均值和中位数那样受重视，但其作用不容忽视。如果你在运营一个客服部，并记录用户反馈的响应时间，那么可能你关心前99%顾客情况的收益远大于只关心中位数个数顾客情况。如果数据形状比较奇怪，只关心平均数个数的顾客情况就更不能反映整体情况了。

标准差和方差

从某种意义上说，均值与中位数都是一组数的“中间的数”：中位数，顾名思义，就是一组数据的中心位置，而均值实际上则是列表中所有数值加权之后的中心。

不过，对于一份数据而言，它的集中趋势可能只是你关心的一个方面。了解通常数据与期望值有多大偏移也同样重要，这称为数据的散布。定义数据的范围有很多方式，比如前面提到的range函数：数据范围由min（最小）值和max（最大）值决定。但是要合理地定义散布程度，还需要以下两个因素：

- 散布程度覆盖的应该只是大多数数据，而非全部数据。
- 散布程度不应该完全由数据的两个极端值决定，这两个极值通常只是异常值而无法代表数据集整体情况。

min（最小值）和max（最大值）通常都是异常值，因此用它们来定义散布程度相当不稳定。换个角度思考，假设我们保持这两个极值不变而改变其余数据，那会怎样？实际上你可以随意地去除其余数据而仍然保持最大值和最小值不变。也就是说，不论你是有200万个还是两个数据点，建立在最大值和最小值基础上的定义都只依赖于数据集的两个数据点。因为我们不能相信任何对大部分数据点都不敏感的摘要，所以要用更好的方式来定义数据集散布程度。

对数据集进行数值摘要已有很多可行的方法。例如，可以计算包含50%数据的范围是什么，围绕中位数的数是什么。用R很容易统计出这些信息：

```
c(quantile(heights, probs = 0.25), quantile(heights, probs = 0.75))
```

或者可把范围扩大，找一个覆盖95%数据的范围：

```
c(quantile(heights, probs = 0.025), quantile(heights, probs = 0.975))
```

这些的确可以很好地衡量数据的散布程度。当你用到更高深的统计学方法时，此类范围

定义方法比比皆是。但是历史上的统计学家们却用过一个不太一样的散布程度衡量方法：该方法明确地定义为方差（variance）。大致来说，这个定义是为了衡量数据集里面任意数值与均值的平均偏离程度。作为一名黑客，我们要写一个自己的方差计算函数，而不是写方差的数学计算公式：

```
my.var <- function(x) {
  m <- mean(x)
  return(sum((x - m) ^ 2) / length(x))
}
```

和之前一样，把这个函数与R的var函数作比较，以确保代码正确：

```
my.var(heights) - var(heights)
```

实现的函数和R内置函数var的执行结果有偏差。从理论上，可以找一些理由来解释：即使不考虑浮点运算的精度问题，仍然还会有偏差。事实上，另一个重要原因是函数的实现方式与R内置函数所用的方法不同：在正规的方差定义中，除数并不是向量的长度，而是向量的长度减1。之所以这样做，是因为从经验数据估算的方差会由于一些细微原因比其真值要略小。要修补这个误差，假设数据集有n个数据点，你会自然地想到用比例因子n/(n-1)与方差估计值相乘，由此更新后的my.var函数实现：

```
my.var <- function(x) {
  m <- mean(x)
  return(sum((x - m) ^ 2) / (length(x) - 1))
}
my.var(heights) - var(heights)
```

用这个版本的my.var函数计算方差，可以和R内置方差估算函数的结果完全一致。上文关于浮点运算有偏差的观点在我们进行长向量运算时很常见，但是本例中的向量长度还不涉及此问题。

用方差衡量数据集散布程度合乎情理，但是它的值几乎比数据集中任何一个值都要大很多。用一个很直接的方法就可以看到这个现象，看看与均值相差正负单位方差之间的数值：

```
c(mean(heights) - var(heights), mean(heights) + var(heights))
#[1] 51.56409 81.17103
```

这个范围实际上比源数据集的范围要大：

```
c(mean(heights) - var(heights), mean(heights) + var(heights))
#[1] 51.56409 81.17103
range(heights)
#[1] 54.26313 78.99874
```

之所以目前的数据范围超出原始数据范围，是因为定义方差的方式是衡量列表中数据与均值的平方距离，而没有对其进行开方。为使一切都回归原始的范围，需要用标准差（standard deviation）代替方差，即方差的平方根：

```
my.sd <- function(x) {
  return(sqrt(my.var(x)))
}
```

在进行下一步操作之前，最好检验一下你的实现和R中的内置函数是否一致，此函数在R中叫做sd：

```
my.sd(heights) - sd(heights)
```

因为现在计算的值在正确的范围内，所以有必要重新估算数据范围，看看偏离均值单位标准差的数值：

```
c(mean(heights) - sd(heights), mean(heights) + sd(heights))
# [1] 62.52003 70.21509
range(heights)
#[1] 54.26313 78.99874
```

既然用的是单位标准差，而不是单位方差，那么得到的数据范围就轻松落入极差（range）范围内。要对数据内部集中程度有个基本认识，方法之一便是对比基于标准差的范围和基于分位数的范围：

```
c(mean(heights) - sd(heights), mean(heights) + sd(heights))
# [1] 62.52003 70.21509
c(quantile(heights, probs = 0.25), quantile(heights, probs = 0.75))
#25%          75%
#63.50562     69.17426
```

用quantile函数可以看到，大致有50%的数据落在距离均值正负一个标准差之间的范围之内。这是个典型的数据特征，对这份身高数据来说更是如此。但是要最终精确地刻画数据的形状，还需要对数据进行可视化操作，并定义一些正式用语来描述数据的形状。

可视化分析数据

计算数据的数值摘要的意义毋庸置疑，这毕竟是经典统计学的核心。但是对于很多人来说，数字并不能有效地传递出他们想看到的信息。要发现数据中的模式有一个更有效的方法，那就是使数据可视化。本节会介绍两个最简单的可视化数据分析方法：一种是单列可视化，它侧重数据的形状；另一种是双列可视化，它侧重两列之间的关系。除了介绍数据可视化工具外，我们还将介绍几种标准数据形状，当你拿到新数据时就可以试着与这些标准形状比对一下。这些理想的形状，又称为分布，是统计学家们研究了很多年的标准模式。如果你发现自己的数据符合这些形状之一，那么通常可以对数据有个大致

推断：数据源如何，有何大致属性，等等。甚至当你的数据只是近似符合这些形状，也可以用它们作为基本分布形状，叠加出更复杂也更加逼近数据的形状。

言归正传，接下来开始对之前一直在操作的身高和体重数据进行可视化。这实际上是一份相当复杂的数据，我们多次用它阐释了本书中提出的理念。人们用到的最典型的单列可视化方法就是直方图。该方法首先把数据集分放到若干个区间里，然后统计每个区间里数据的条数。如图2-5所示，以1英寸为区间宽度创建直方图来可视化身高数据，R代码如下：

```
library('ggplot2')
data.file <- file.path('data', '01_heights_weights_genders.csv')
heights.weights <- read.csv(data.file, header = TRUE, sep = ',')
ggplot(heights.weights, aes(x = Height)) + geom_histogram(binwidth = 1)
```

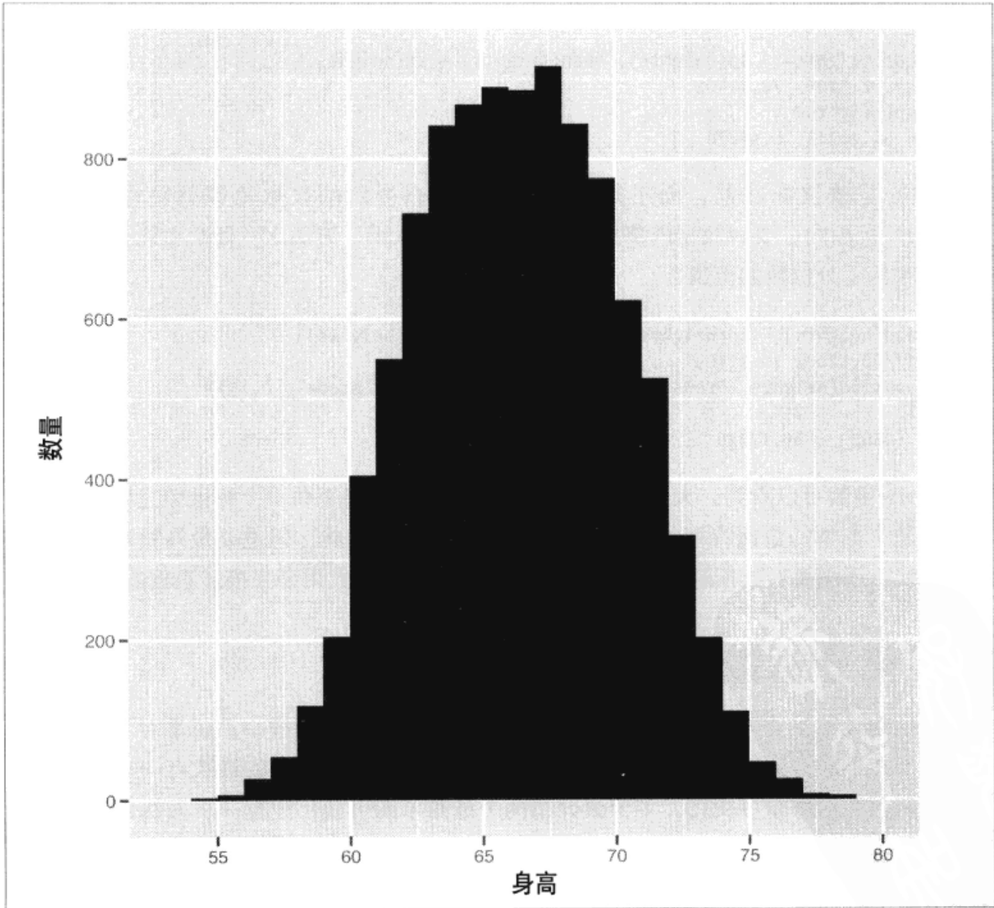


图2-5：10 000人的身高直方图

你马上就会发现：数据呈钟形。大部分数据处于中间，与均值和中位数接近。但这只是因所选的直方图类型而造成的假象。检验是否存在假象的方法之一就是尝试不同的区间宽度。在你使用直方图的时候要始终牢记一点：区间宽度是你强加给数据的一个外部结构，但是它却同时揭示了数据的内部结构。在构建直方图的时候，如果设置了错误的参数，那么你发现的模式（即使真实存在）也会轻易地消失。用5英寸的区间宽度重新构建直方图，如图2-6所示，代码如下：

```
ggplot(heights.weights, aes(x = Height)) + geom_histogram(binwidth = 5)
```

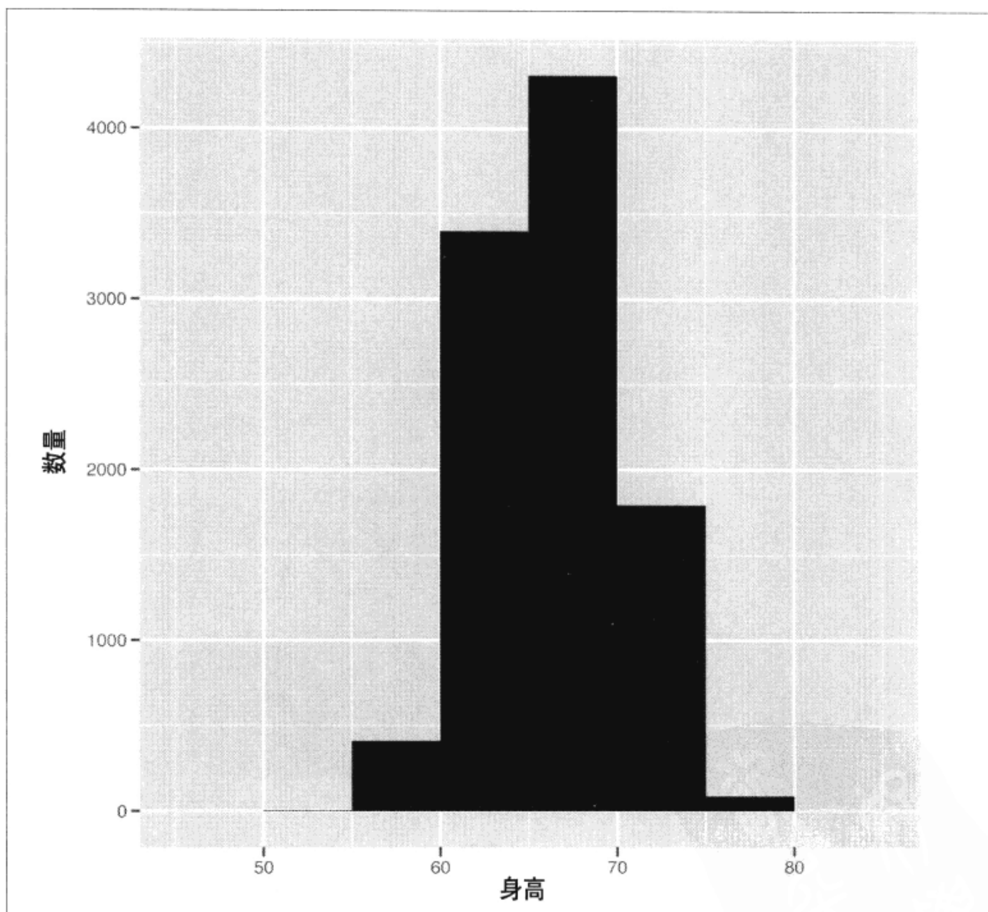


图2-6：10 000人的身高直方图

当采用一个较大的区间宽度值时，数据的很多结构就不见了。虽然还有个顶峰，但是之前看到的对称性已经几乎不存在了。这叫做过平滑（oversmoothing），与之相反的问题

则称为欠平滑（undersmoothing），也是很危险的。再次调整区间宽度值，这一次是一个非常小的值：0.001英寸，如图2-7所示：

```
ggplot(heights.weights, aes(x = Height)) + geom_histogram(binwidth = 0.001)
```

这一次使数据欠平滑了，因为采用了一个相当小的区间宽度值。因为数据量很大，所以从这个直方图中依然可以发现一些有价值的东西，但如果是一份只有100个数据点的数据集，你却用了这种区间宽度，基本上一文不值。

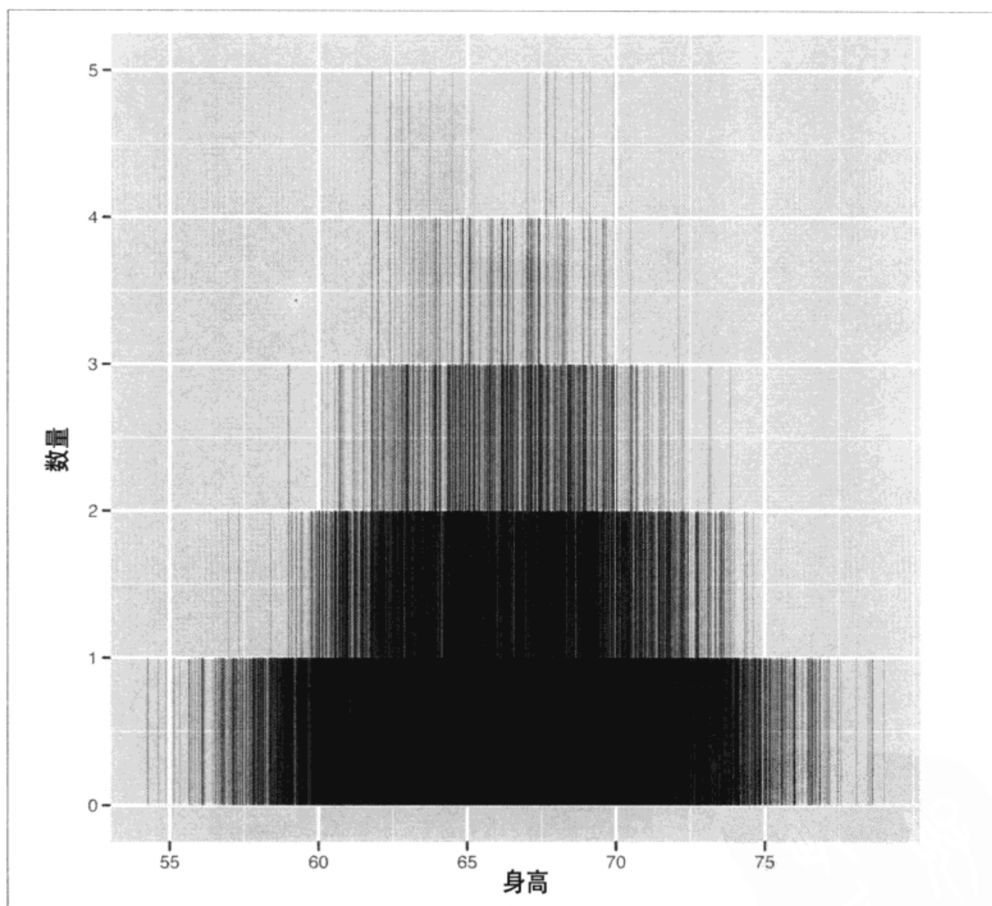


图2-7：10 000人的身高直方图

调整区间宽度值是一件枯燥的事，而且即便是最佳的直方图，在我们看来其锯齿状也很明显，因此我们倾向于选择一种和直方图类似的方法来可视化，即核密度估计（Kernel Density Estimate，KDE）或者叫做密度曲线图（density plot）。尽管密度曲线图也无法规避直方图令人纠结的欠平滑和过平滑问题，但是我们首先考虑的是美观——密度曲线

图尤其在大数据集上更接近我们所期望的理论形状。此外，密度曲线图也有一些理论优势：揭示数据潜在的形状，密度曲线图需要的数据点比直方图要少。而且，生成密度曲线图和直方图一样简单。如图2-8所示，构建了身高数据的第一个密度曲线图：

```
ggplot(heights.weights, aes(x = Height)) + geom_density()
```

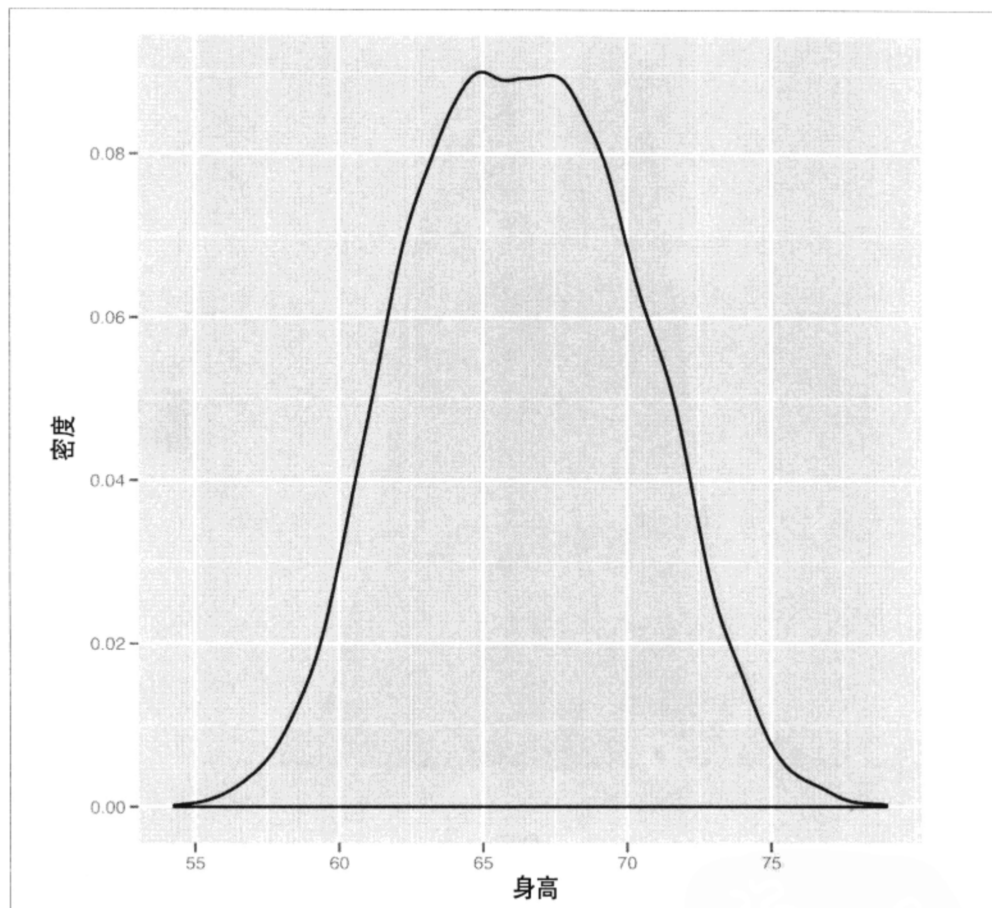


图2-8：10 000人身高数据密度曲线图

密度曲线图的平滑性有助于我们发现数据的模式类型，而这很难在直方图中发现。从密度曲线图中看到，峰值处竟然有些平坦，不禁令人生疑。因为期望的标准钟形曲线在峰值处不是平坦的，这让我们很想知道是不是在数据集的峰值处还隐藏着更多的结构。当你觉得可能有些结构缺失的时候，有一个方法可以试一试：根据其中任意一个定性变量将曲线分开。现在，根据数据集当中的性别将曲线分离成两部分。在图2-9中，构建一

个密度曲线图，它由两个密度曲线叠加而成，但已经通过颜色标明了其所代表的不同性别：

```
ggplot(heights.weights, aes(x = Height, fill = Gender)) + geom_density()
```

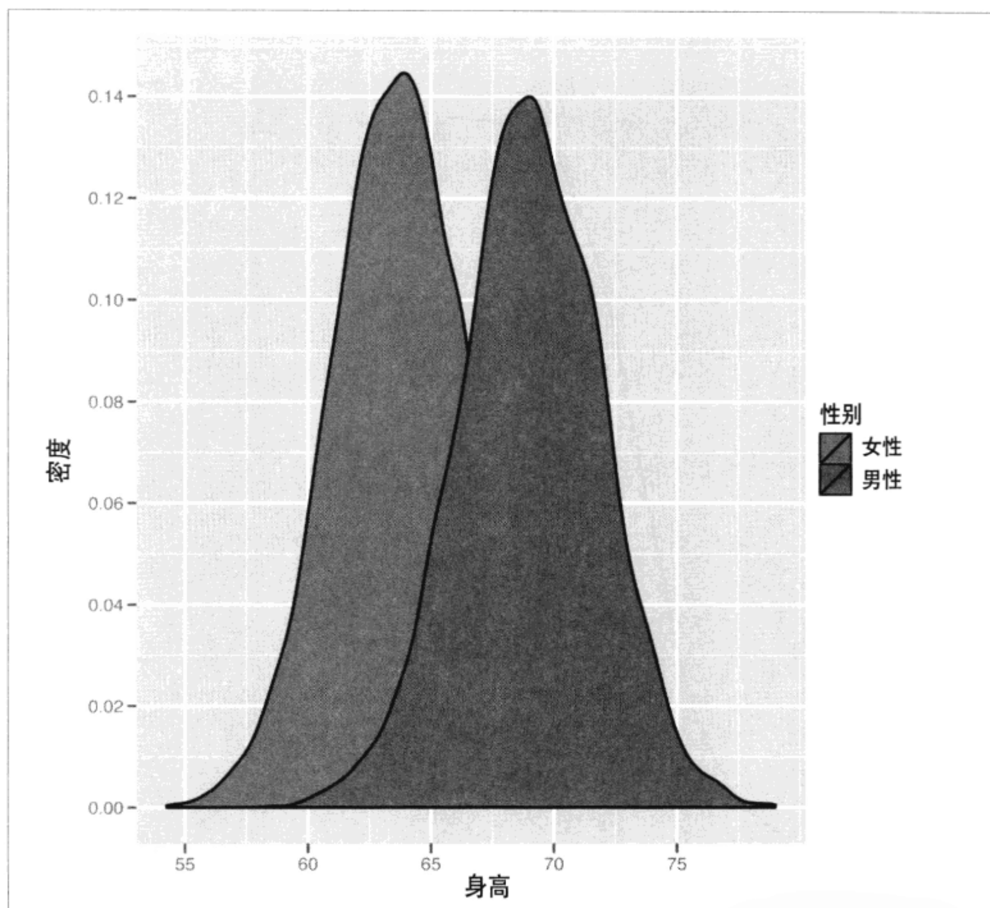


图2-9：与性别相关的10 000人身高数据密度曲线图

在这张图里我们一下就发现了此前忽视的一个隐藏模式：我们虽然没有看到一个钟形曲线，却看到了两个部分重叠的钟形曲线。这并不意外，因为男性和女性的平均身高是不一样的。我们可能希望两个性别的体重数据曲线也是与此相同的钟形曲线结构。如图2-10所示，给数据集的体重数据构建一个新的曲线图。

我们再一次看到了两个钟形曲线结构的混合。在本书余下内容里，还会比较详细地探讨这种混合的钟形曲线，但是有必要现在就给这种结构命个名：混合模型，它是由两个标准分布混合而形成的一个非标准分布。

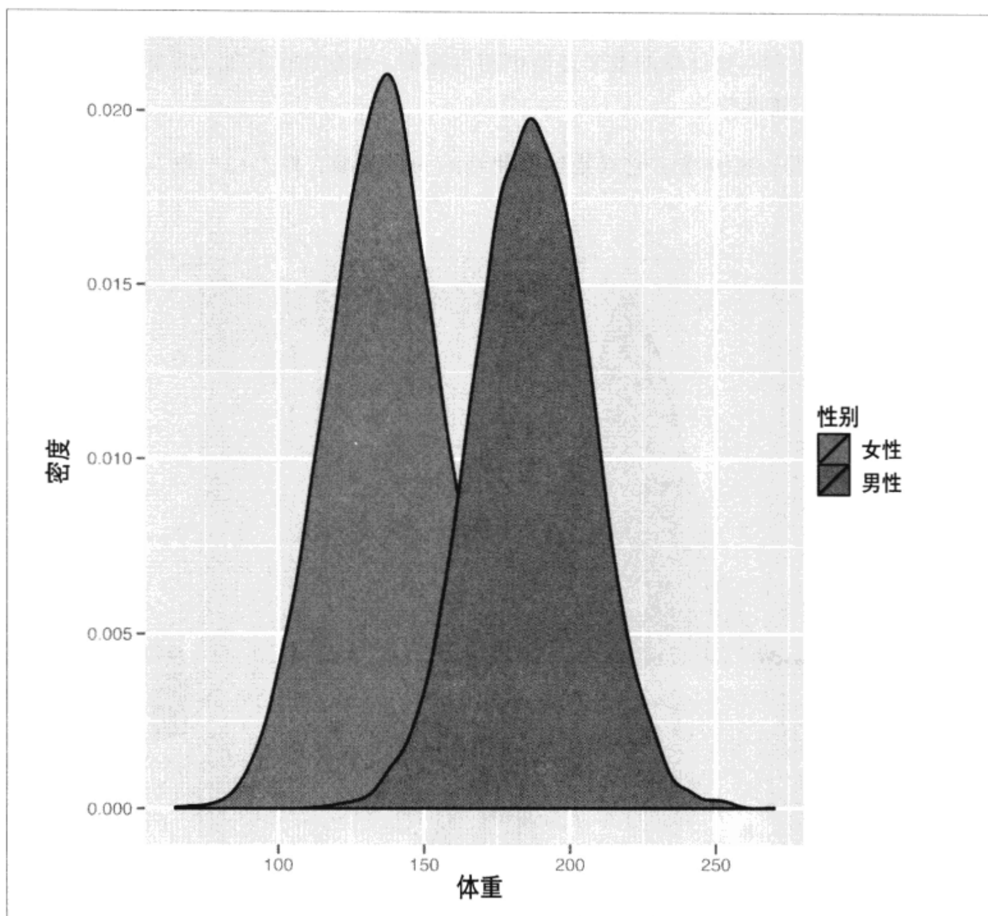


图2-10：与性别相关的10 000人体重数据密度曲线图

当然，我们需要把标准分布讲得更清楚些，这样才能清晰明白上面那句话的意义，因此从最理想的数据分布入手：正态分布，也称为高斯分布或钟形曲线。正态分布的例子有很多，上面的混合分布就是由两个正态分布组合而成的，这里的每一个正态分布叫做一个分片（facet）。在图2-11中，把此前展示的密度曲线分片，以便读者能看到两个单独的钟形曲线。在R中，可以用下面的代码实现这种分片：

```
ggplot(heights.weights, aes(x = Weight, fill = Gender)) + geom_density() +
facet_grid(Gender ~ .)
```

分片完成后，两个钟形曲线清晰可见，一个是中心在137.5磅的女性钟形曲线图，另一个是中心在187.5磅的男性钟形曲线图。这种类型的钟形曲线就是正态分布。这个形状十分常见，以至于我们下意识觉得“正态”才是数据的“正常形态”。但这是不对的，我们

关心的很多事物，从人们的年收入到股价每日的涨跌，用正态分布来描述都不是特别适合。但是，正态分布在统计学的数学理论中相当重要，因此相比其他大多数分布，关于正态分布的研究更透彻些。

从更抽象的层面看正态分布，它只是钟形曲线的一种类型，图2-12～图2-14都是钟形曲线。

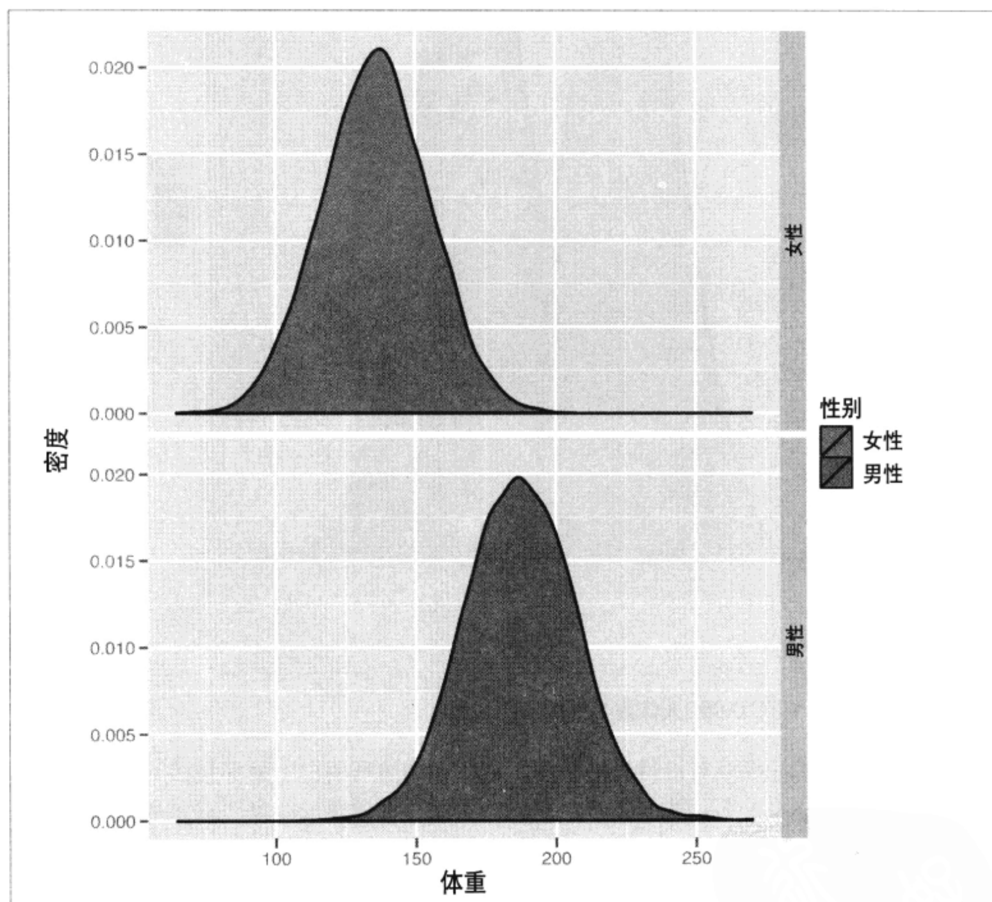


图2-11：10 000人体重数据密度曲线图，以性别分片（单位：磅）

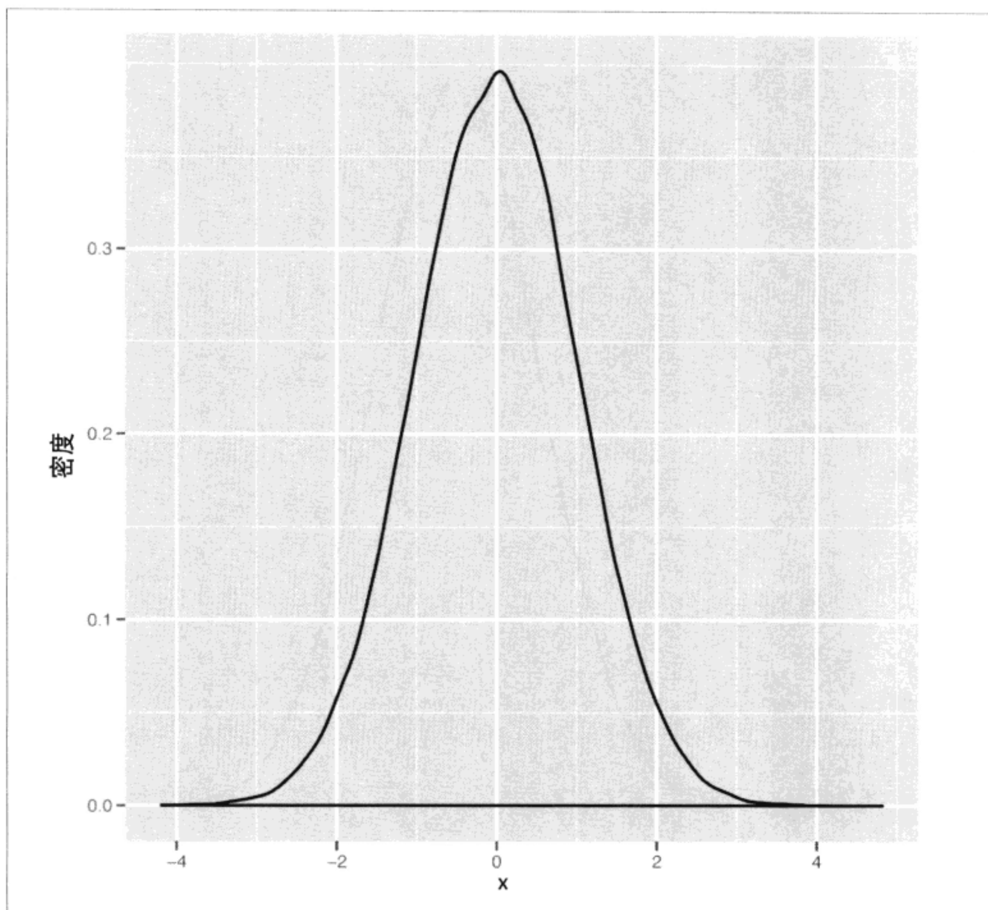


图2-12：均值为0，方差为1的正态分布

在这些图中，有两个变量：分布的均值，它决定钟形曲线的中心所在；分布的方差，它决定钟形曲线的宽度。可以通过下面的代码得到不同的钟形曲线图，调整其中的参数，直到最终的钟形曲线看起来很舒服。调整时修改其中m和s的值即可：

```
m <- 0
s <- 1
ggplot(data.frame(X = rnorm(100000, m, s)), aes(x = X)) + geom_density()
```

用这段代码生成的曲线的基本形状是一样的；改变m和s只是移动其中心或者伸缩其宽度。读者从图2-12～图2-14中可以看到，曲线的具体形状在改变，但是其整体轮廓并没有变化。要注意的是，钟形并不是判断数据是否为正态分布的充分条件，因为还存在其他钟形分布，稍后会介绍其中一种。利用正态分布，可以定义一些关于数据形状的定性概念，所以接下来让我们快速了解一些术语。

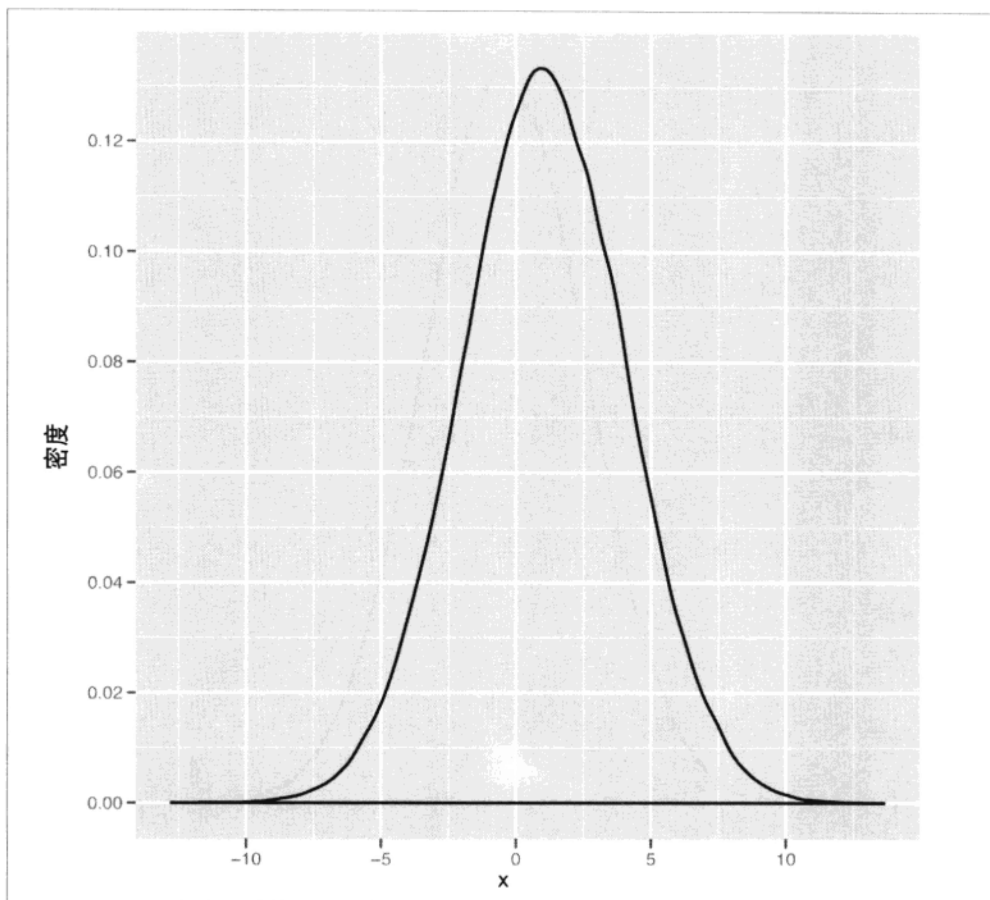


图2-13：均值为1，方差为3的正态分布

首先，重新讨论到现在都还被我们冷落在一旁的众数。之前提到过，连续数值列表的众数不好定义，因为没有数值重复出现。但是，连续数值的众数却能用可视化的方法解释清楚：当构建一条密度曲线时，数据的众数就在钟形的峰值处。举个例子，如图2-15所示。

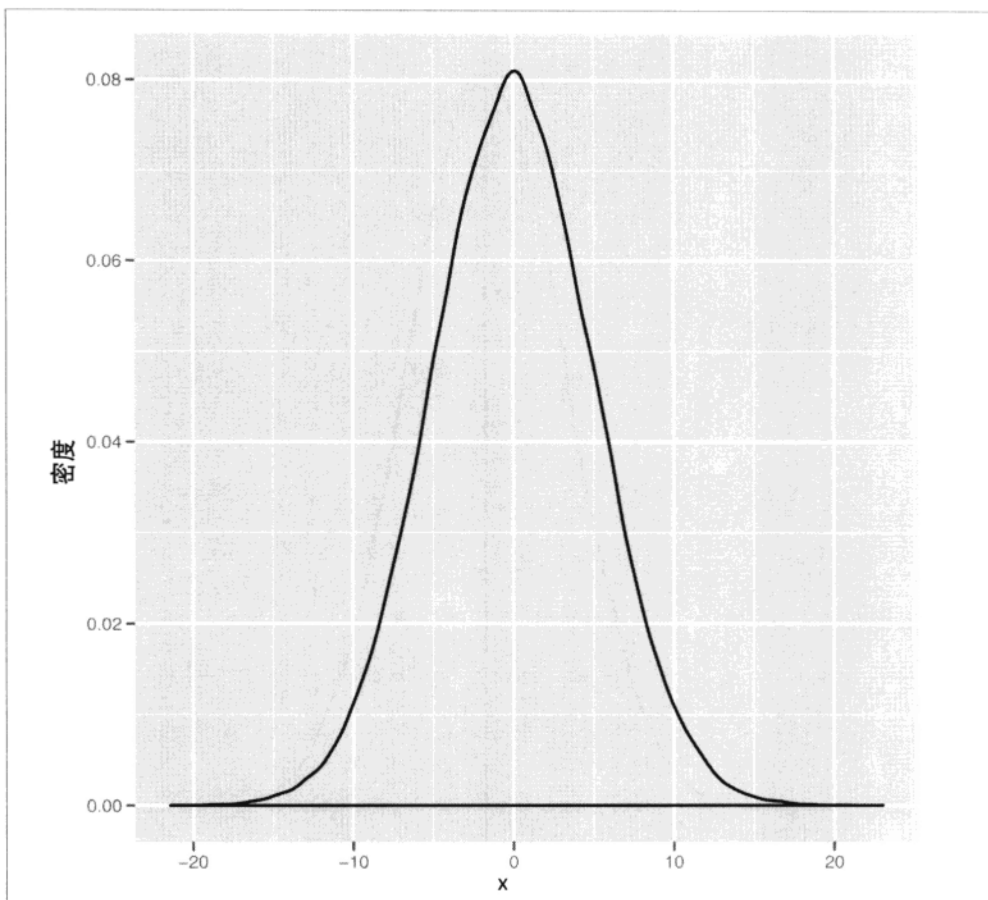


图2-14：均值为0，方差为5的正态分布

用可视化方法估计众数，密度曲线图比直方图要容易得多，这也是我们推荐使用密度曲线图的原因所在。当你看到密度曲线图，众数的意义一目了然，然而直接面对数字的时候，几乎看不到它背后的含义。

既然已经定义了众数，就应该指出正态分布所定义的众数有一个特点：它只有一个众数，同时也是数据的均值和中位数。作为对比例子，图2-16所示的图有两个众数，图2-17所示的图有三个众数。

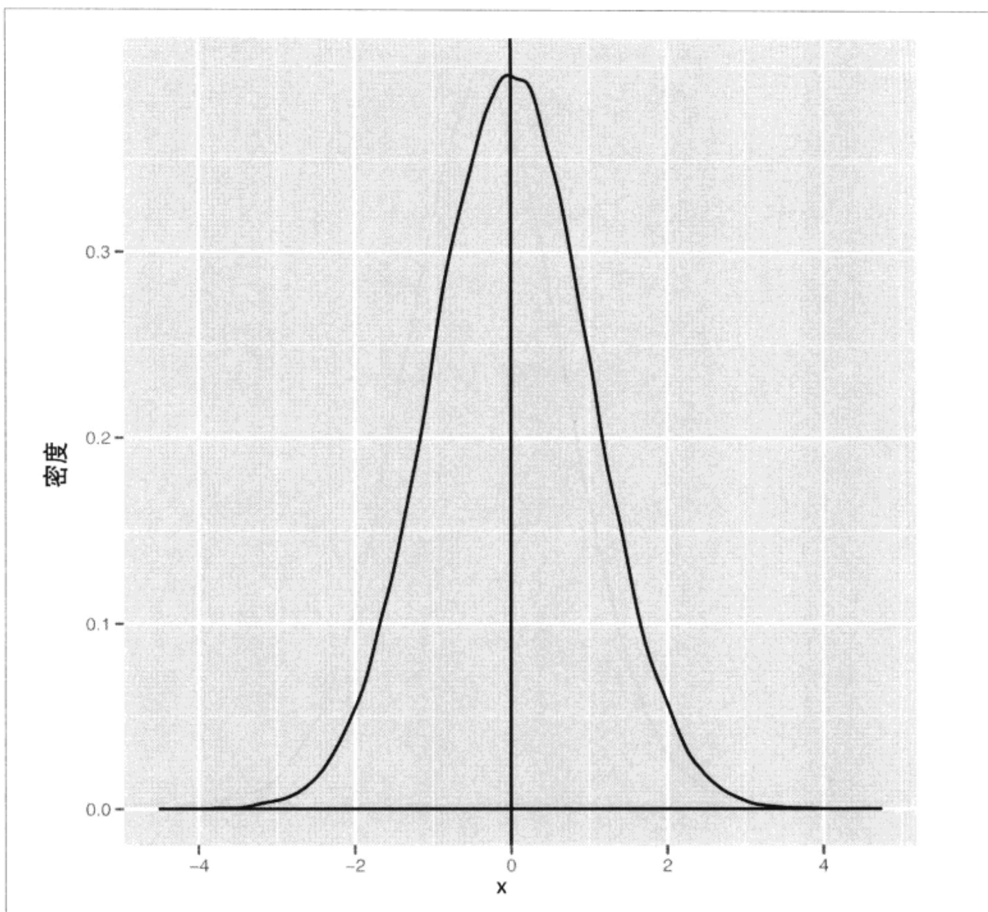


图2-15：标出众数的正态分布

当谈到数据的众数个数时，会用到以下术语：只有一个众数的分布叫单峰（unimodal）；有两个众数的分布叫双峰（bimodal）；有两个以上众数的分布叫多峰（multimodal）。

还可以从一个定性的区别来划分出两类数据，那就是对称分布（symmetric）数据和偏态分布（skewed）数据。图2-18和图2-19所示分别是对称分布和偏态分布。

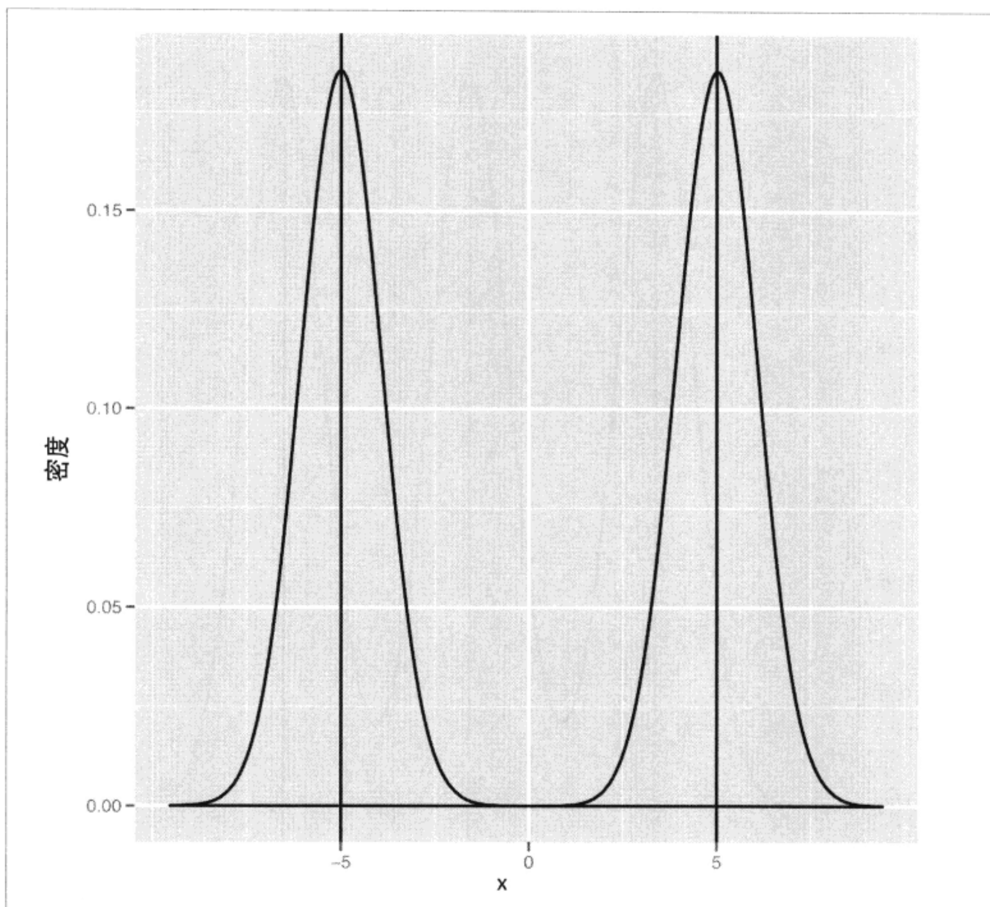


图2-16：两个正态分布混合，两个众数都已标出

对称分布的特点是：图2-18中众数的左右两边形状一样。正态分布就有这个特点，该特点说明观察到小于众数的数据和大于众数的数据的可能性是一样的。对比之下，图2-19所示图形向右偏斜，说明在众数右侧观察到极值的可能性要大于其左侧，这种图形称为伽玛分布（gamma distribution）。

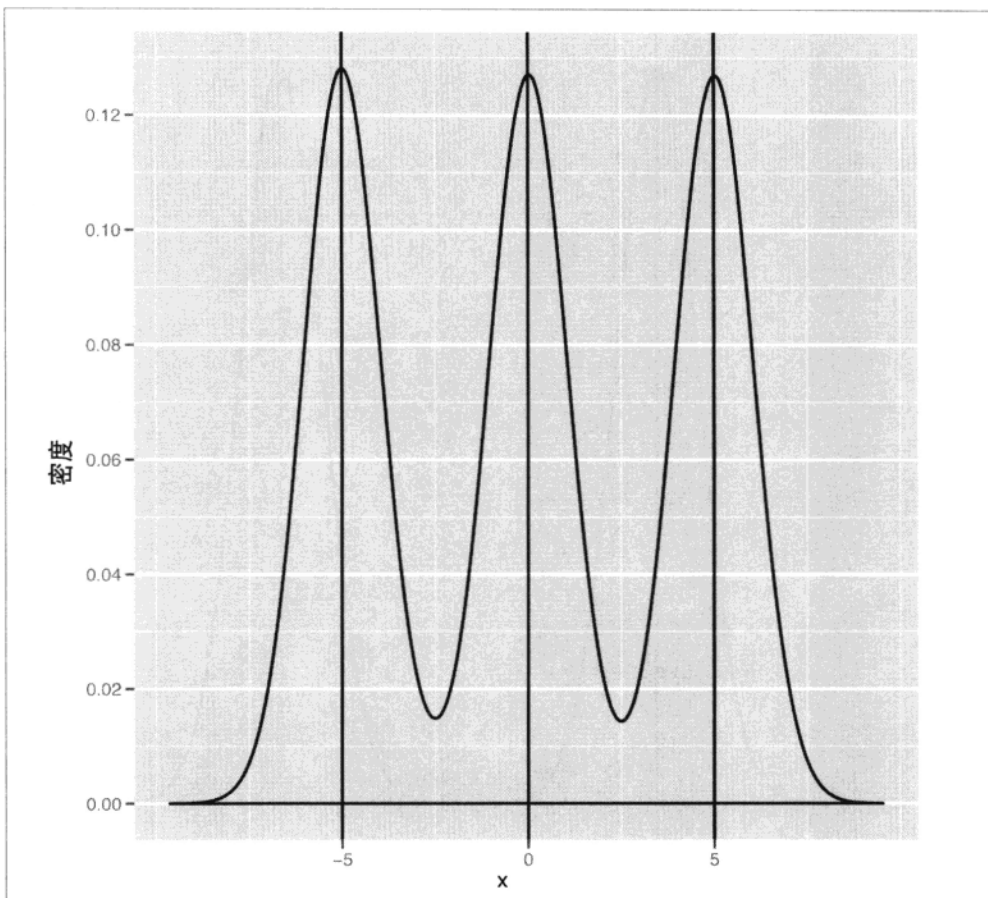


图2-17：三个正态分布混合，三个众数都已标出

最后我们要根据另一个定性的区别来划分出两类数据：窄尾分布（thin-tailed）数据和重尾分布（heavy-tailed）数据。稍后我们会用一张标准的图来说明两者之间的区别，但是这个区别用语言更容易表述。窄尾分布所产生的值通常都在均值附近，99%的可能性都是这样。比如，正态分布在99%的情况下所产生的数据偏离均值都不会超过三个标准差。相比之下，另一个钟形分布——柯西分布（Cauchy distribution）大约只有90%的值落在三个标准差范围内。距离均值越远，这两个分布的特点越不同：正态分布几乎不可能产生出距离均值有六个标准差的值，然而柯西分布仍有5%的可能性。

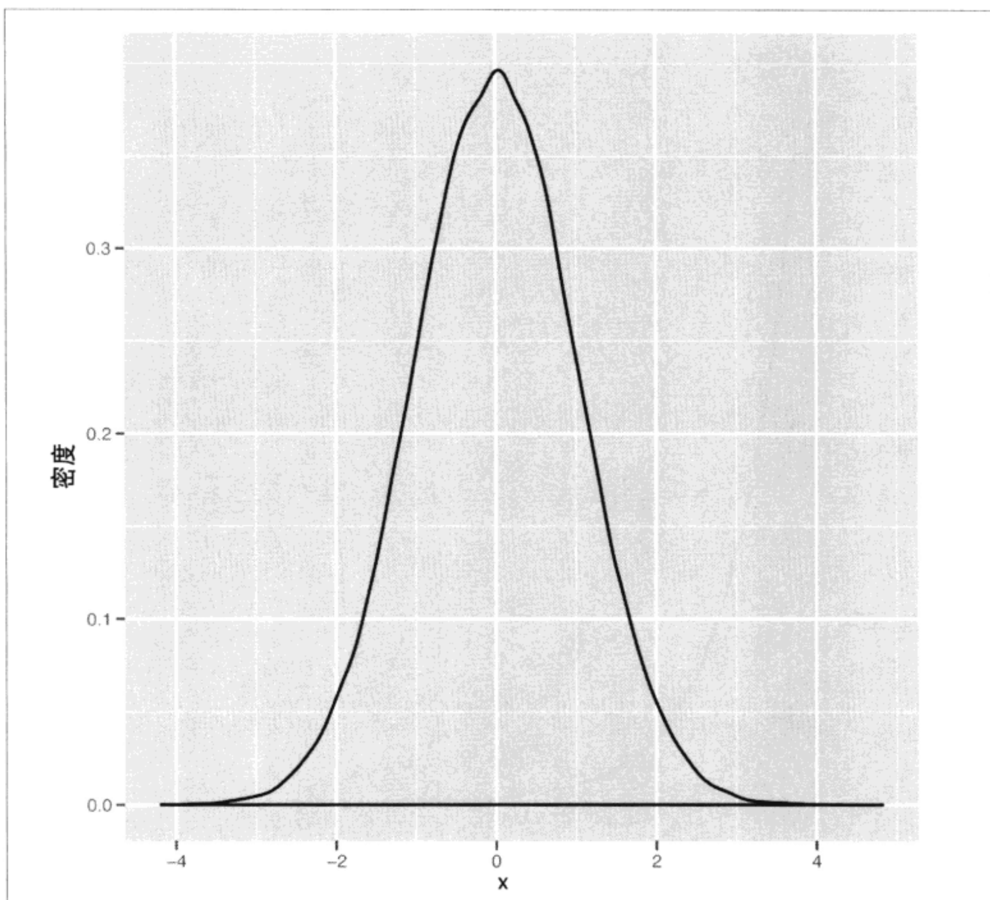


图2-18：对称分布

图2-20和图2-21的对比就是窄尾分布和重尾分布的典型区别。

但是，我们认为通过下面这种方法，你可以理解得更直观：亲自用这两种分布分别产生大量数据，并观察结果。R可以轻松实现这一点，代码如下：

```
set.seed(1)
normal.values <- rnorm(250, 0, 1)
cauchy.values <- rcauchy(250, 0, 1)
range(normal.values)
range(cauchy.values)
```

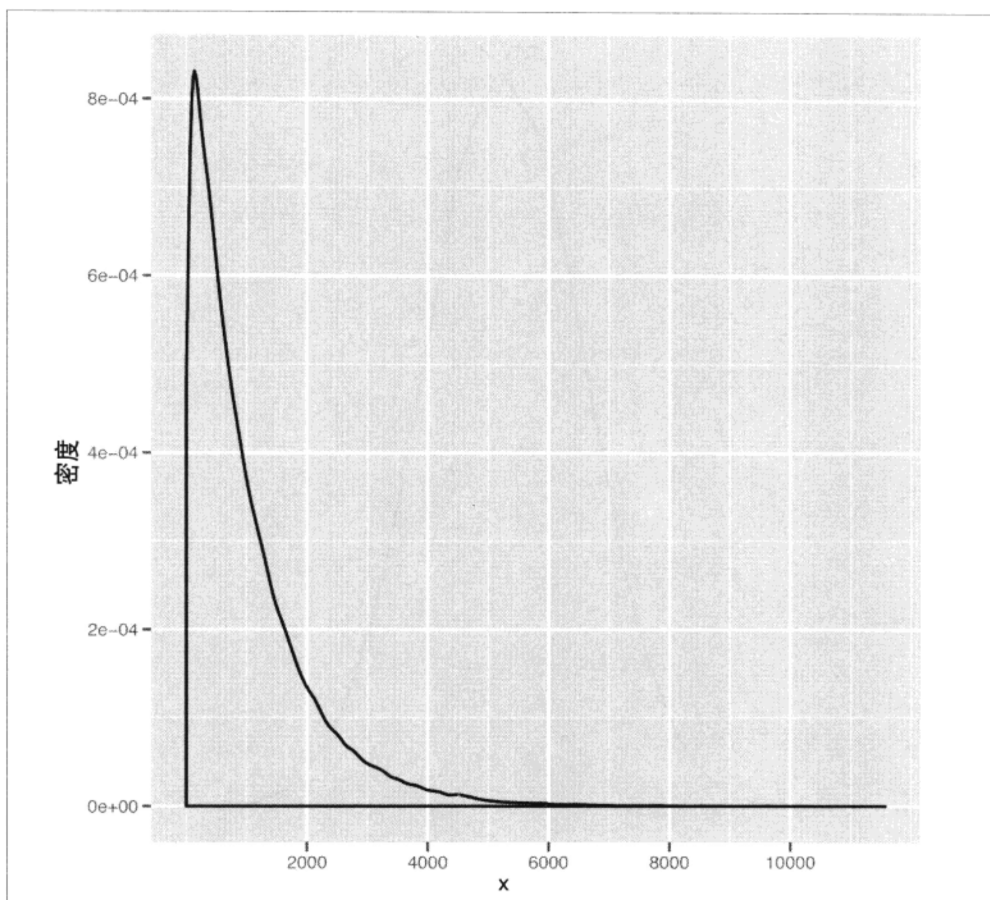


图2-19：偏态分布

把结果画出来可以看得更清楚一些：

```
ggplot(data.frame(X = normal.values), aes(x = X)) + geom_density()
ggplot(data.frame(X = cauchy.values), aes(x = X)) + geom_density()
```

本节就正态分布及柯西分布展开的讨论到此为止，接下来我们来对正态分布的本质特性做个总结：它是单峰的、对称的分布，也是钟形的窄尾分布。柯西分布也是单峰的、对称的分布，也是钟形曲线，却是重尾分布。

本节是关于密度曲线图的，在讨论了正态分布之后，我们再来看看两幅规范的图像，从而为本节画上句号：一个是有点偏斜的伽玛分布，另一个是非常偏斜的指数分布。这两个分布在后面内容中都会用到，因为它们在实际的数据中都出现了。但有必要现在就开始讲解，以便更加直观地了解偏态分布。

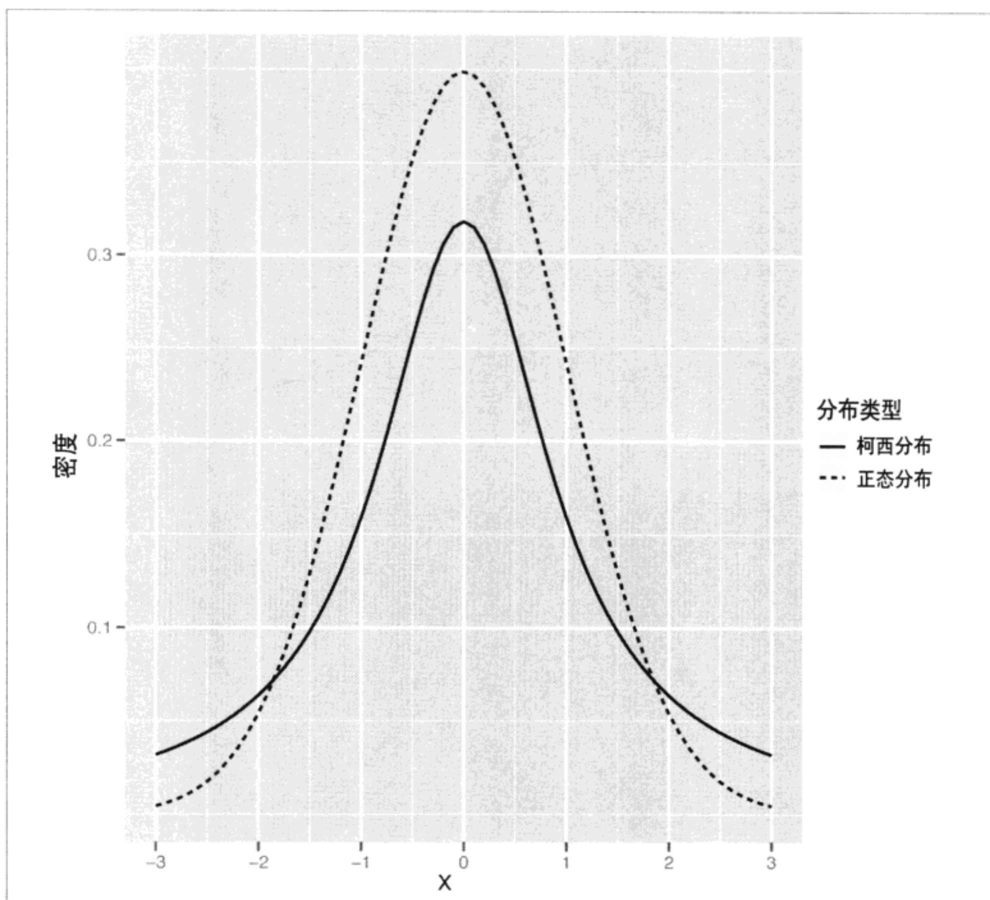


图2-20：重尾的柯西分布和窄尾的正态分布

首先从伽玛分布入手。它很灵活，推荐你自己动手操作一下。下面是示例代码：

```
gamma.values <- rgamma(100000, 1, 0.001)
ggplot(data.frame(X = gamma.values), aes(x = X)) + geom_density()
```

伽玛分布的数据绘图结果见图2-22。

从图2-22可以看出，伽玛分布是向右倾斜的，这意味着中位数和均值有时差距很大。在图2-23中，我们把人们玩苹果手机（iPhone）游戏《屋顶狂奔》（Canabalt）的得分绘制成了密度曲线图。

这份真实的数据与理论上的伽玛分布非常相似。我打赌很多其他游戏得分数据的曲线图也与此相似，因此，要想分析游戏数据，需要有一份特别有用的理论工具。

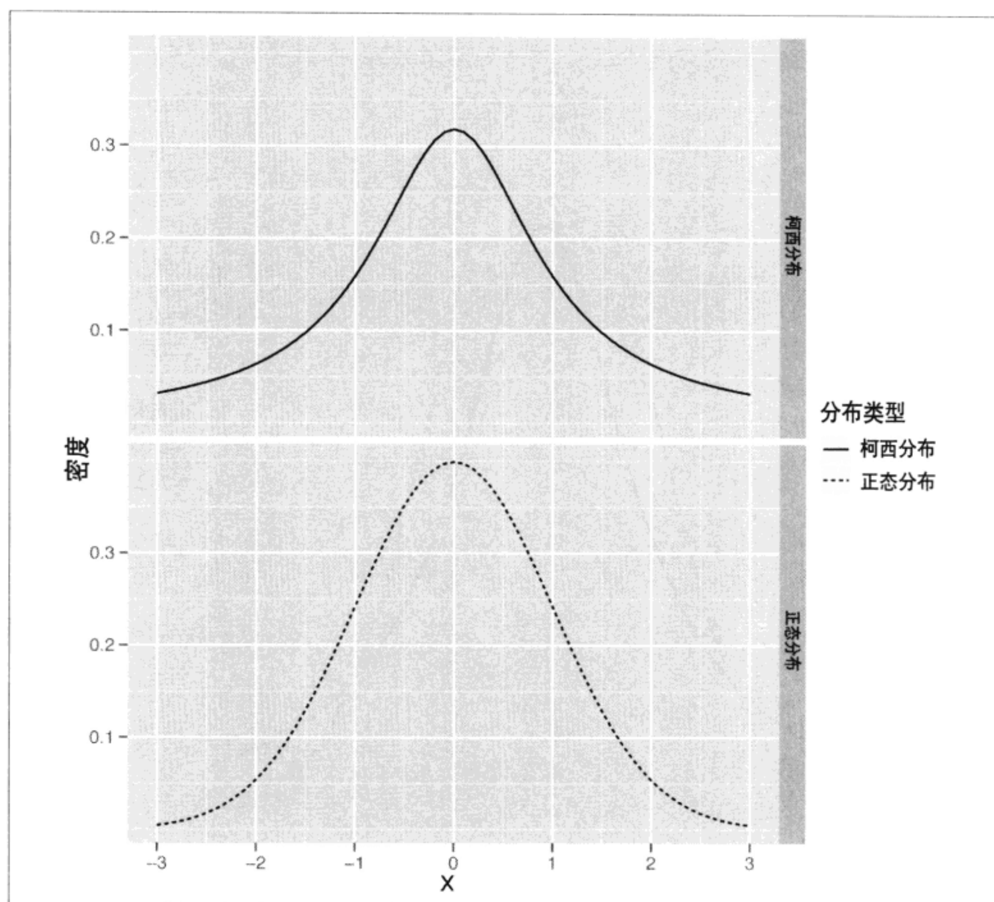


图2-21：分片绘图：重尾的柯西分布和窄尾的正态分布

还有一点需要记住的是：伽玛分布只有正值。在本书后面讲解的随机最优化方法就需要一个全部正值的分布。

最后一个要讲解的分布是指数分布（exponential distribution），它是典型的偏态分布。图2-24是一份满足指数分布的数据。

因为指数分布的众数出现在0值处，所以它特别像是把钟形曲线切掉一半后留下的正值部分。在满足下列条件的情况下常常会出现指数分布：数据集中频数最高的是0，并且只有非负值出现。例如，企业呼叫中心常常发现两次收到呼叫请求的间隔时间看上去符合指数分布。

当你对数据越来越熟悉，对统计学家们所研究的理论分布学习得更深入，你会对这些分布更加熟悉——最重要的原因是：这些分布会反复出现。到目前为止，你从本节学到的是一些简单的定性术语，可以用于向别人描述数据：单峰分布和多峰分布；对称分布和偏态分布；窄尾分布和重尾分布。

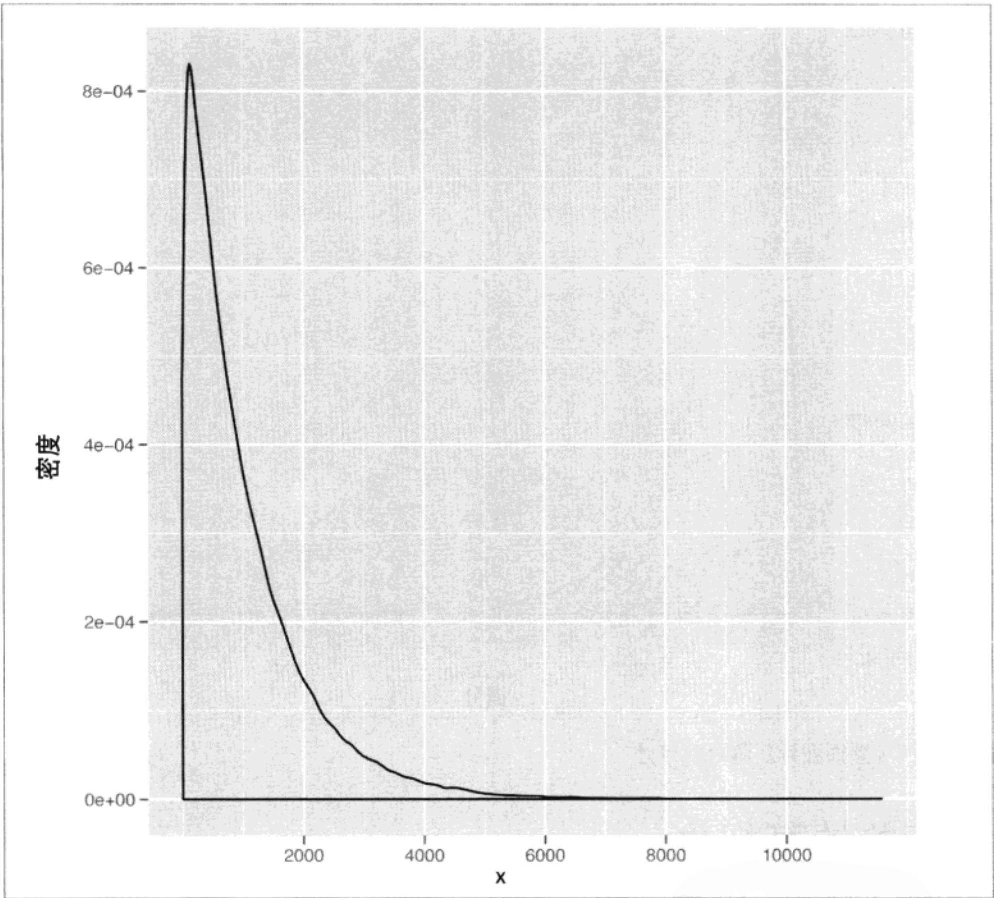


图2-22：偏态分布

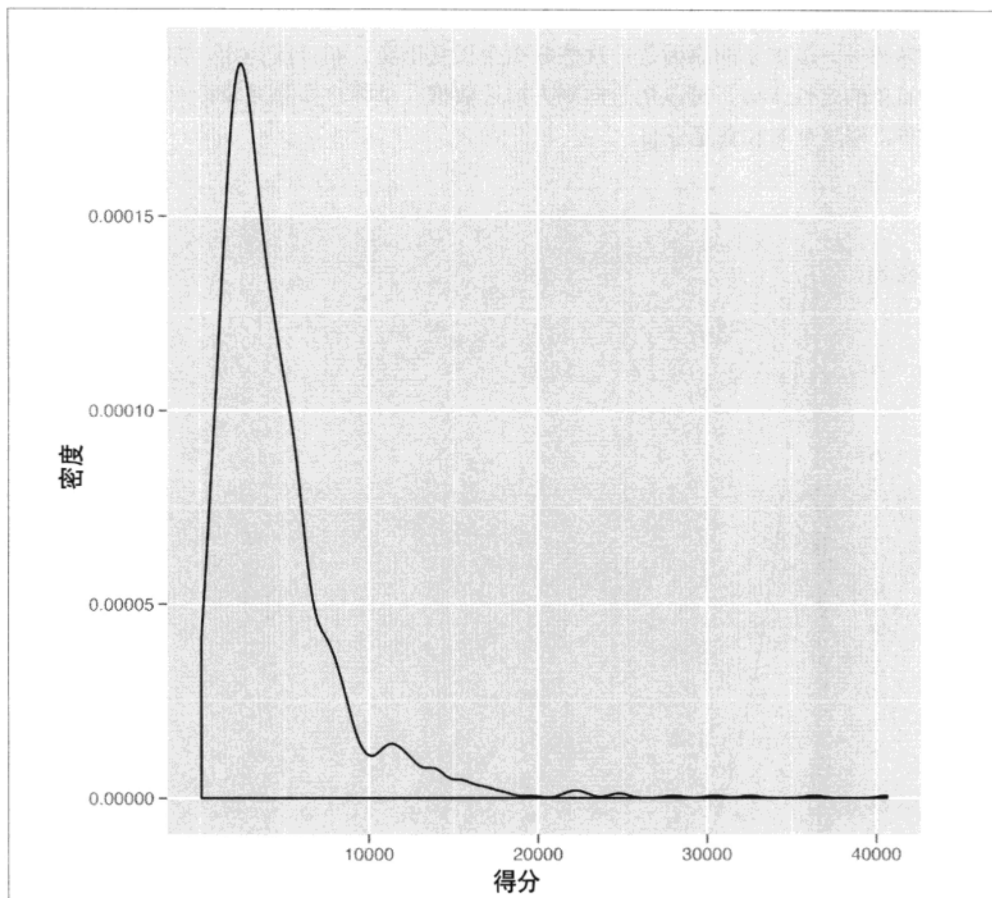


图2-23：《屋顶狂奔》游戏的得分

列相关的可视化

到目前为止，我们只是介绍了数据集中单列处理方法的一些思路。这些方法很有价值：如果在数据集中发现了相似形状，那会让你得到很多信息。发现一个正态分布就说明均值和中位数是相等的，也说明在大部分时候你不会观察到偏离均值超过三个标准差的数值。仅仅学习了一个数据可视化结果，我们就能得到这么多信息。

但是，目前我们所回顾的知识你都能在传统统计学课程中学到，和你热切希望投入其中的机器学习应用还是有所不同。要进行真正的机器学习，我们需要发现数据集中多个数据列之间的关系，以此搞清数据所隐含的意义，而且能够预测未来的一些东西。在本书中我们要接触到的预测问题有如下几个：

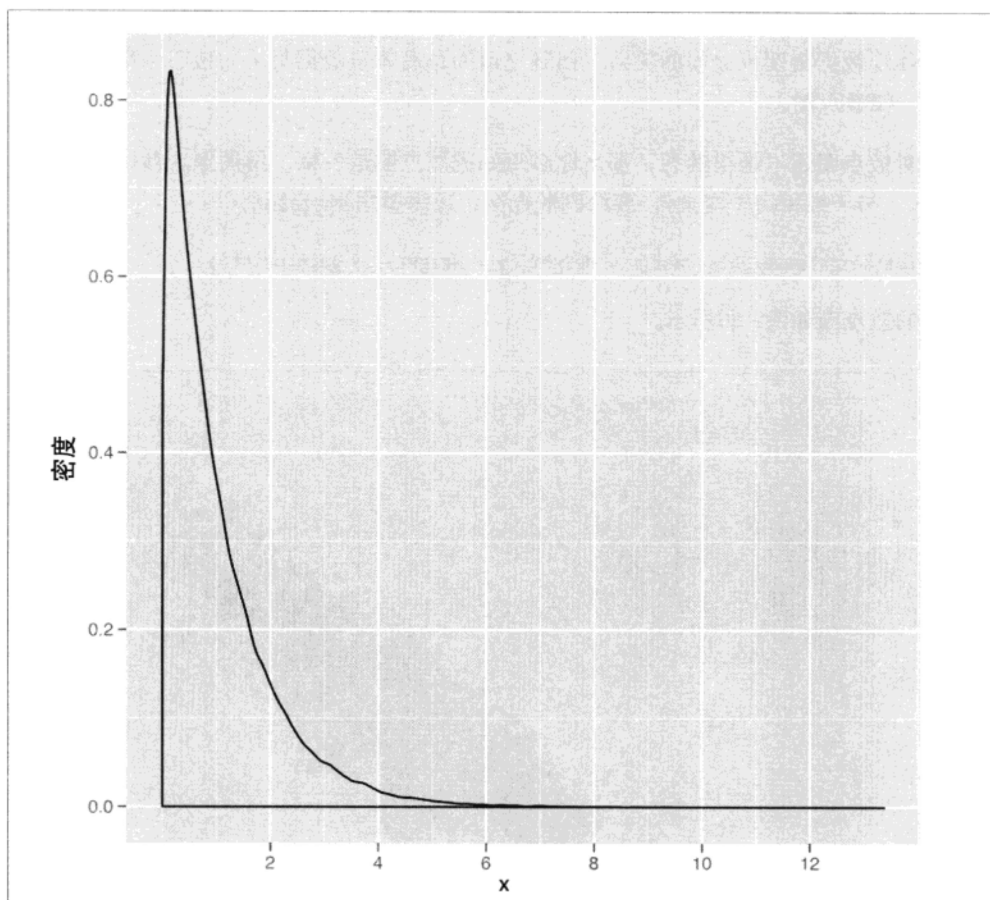


图2-24：指数分布

- 根据一个人身高预测其体重；
- 根据电子邮件文本预测一封邮件是不是垃圾邮件；
- 预测一个人是不是想买此前从未向其推荐过的一件产品。

这些问题仍然可以分成两类：回归问题，要预测的是数值，比如体重，已知的是一组其他数值，比如身高；分类问题，就是给数据贴上标签，比如“垃圾”，已知的是一组数值，比如类似“伟哥”（viagra）和“西力士”（cialis）这些垃圾词汇的词频。本书余下内容中很多地方都会介绍回归和分类的方法，在这里我们希望读者谨记两种数据可视化方法。

第一个就是老套的回归图。在回归图中，我们要画出数据的散点图，观察是否有隐含的形状来体现数据集两列之间的关系。回到之前的身高体重数据集上，接下来给身高和体重绘制一幅散点图。

如果你对散点图还不是很熟悉，那么你必须知道散点图是一幅二维图像，其中一维相当于变量x，另一维相当于变量y。要绘制散点图，还需要用到ggplot。

```
ggplot(heights.weights, aes(x = Height, y = Weight)) + geom_point()
```

所绘制的散点图如图2-25所示。

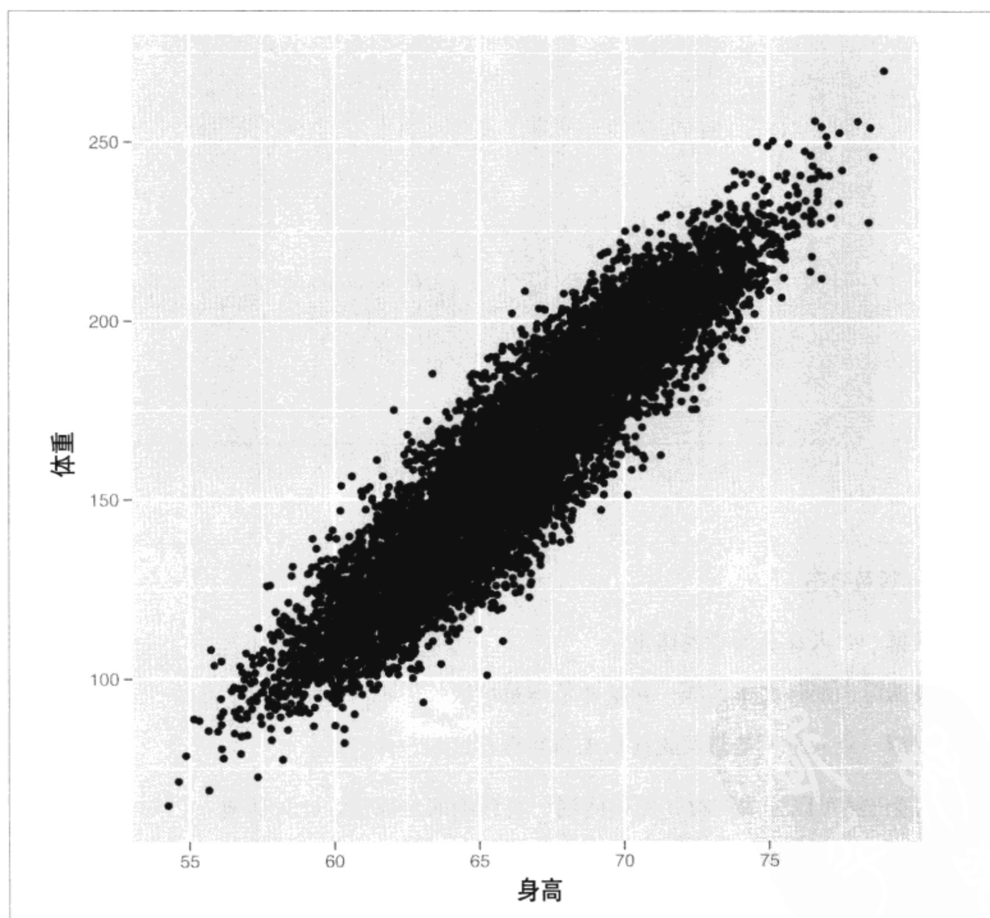


图2-25：身高和体重散点图

从图2-25中可以明显地看到身高和体重之间存在某种关系模式：较高的人会较重。这和直觉明显符合，后面内容将会介绍用于发现这类模式的通用方法。

为了更仔细地研究这个模式，用ggplot2中的平滑方法把所看到的这个线性模式形象地描绘出来：

```
ggplot(heights.weights, aes(x = Height, y = Weight)) + geom_point() + geom_smooth()
```

如图2 26所示，这是一个叠加了平滑模式的新散点图。

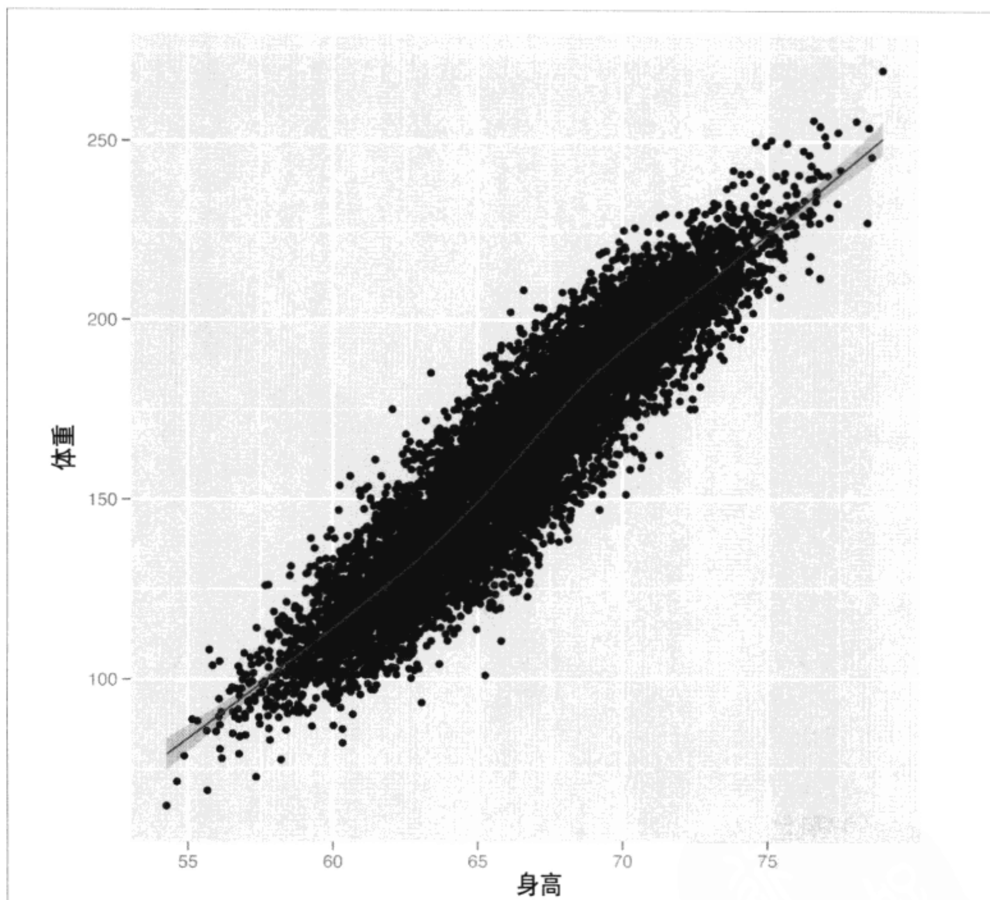


图2-26：带平滑线性拟合的身高和体重散点图

geom_smooth函数可以根据人们的身高预测其体重。在本例中，预测的趋势是一条简单的曲线，如图2-26中的浅灰色标注。曲线周围阴影区是体重预测值的范围，当数据量增加之后，这样的猜测会更准，阴影区也会缩小。因为我们已经用到了所有的数据，所以要体会这种效果最好的方法就是反其道而行之：移除一些数据，然后观察其中的模式如何变得越来越不明显。实验的结果见图2-27~图2-29。

```
ggplot(heights.weights[1:20,], aes(x = Height, y = Weight)) + geom_point() +geom_smooth()
ggplot(heights.weights[1:200,], aes(x = Height, y = Weight)) + geom_point() +geom_smooth()
ggplot(heights.weights[1:2000,], aes(x = Height, y = Weight)) + geom_point() +geom_smooth()
```

回想一下，如果用一个数据列去预测另一个数据列，且要预测的是数值，那么就称为回归。相反，若要预测的是标签，就称为分类。而对于分类，你应该知道，和图2-30差不多。

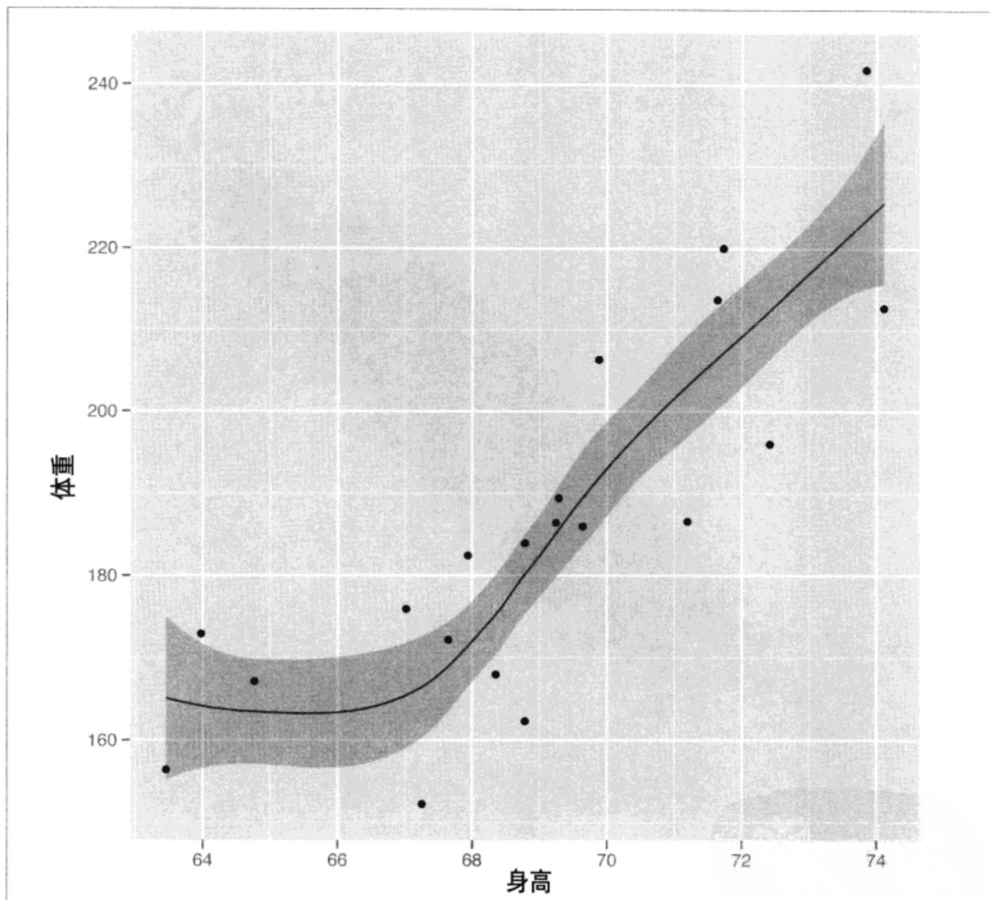


图2-27：20份数据的身高和体重散点图

在这张图中，用所有数据绘出了每个人的身高和体重，也用颜色标注了每个数据点的性别。从而清楚地看到数据集有两个不同的群体。要用ggplot2来绘制这幅图，需要运行下面的代码：

```
ggplot(heights.weights, aes(x = Height, y = Weight, color = Gender)) + geom_point()
```

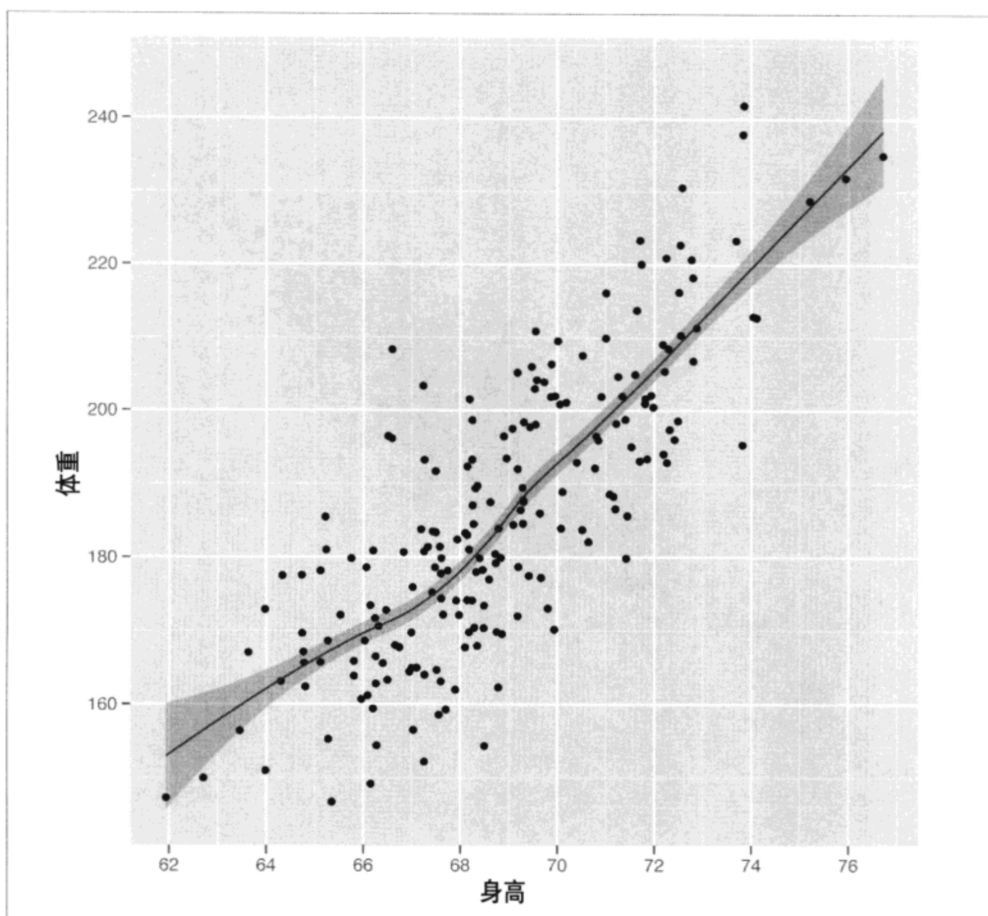


图2-28：200份数据的身高和体重散点图

这张图是标准的分类图。在分类图中，用数据生成散点图，然后用第三列给不同标签的数据点标上颜色。对于身高和体重数据，加入的第三列是数据中每个人的性别。再看这幅图，似乎可以只用身高和体重来猜这个人的性别。分类要做的事就是利用其他数据来预测诸如性别这样的分类变量，第3章会比较详细地讲解分类的算法。而现在，我们只简单看看运行一个标准的分类算法之后的结果，如图2-31所示。

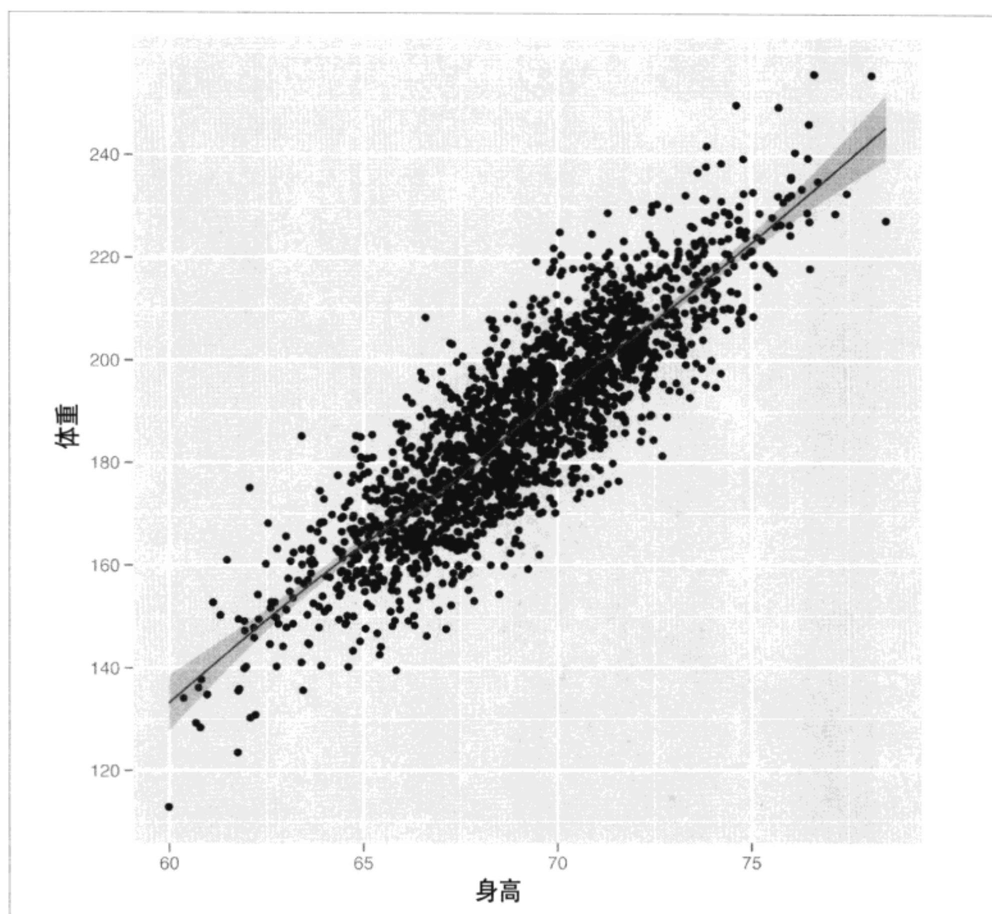


图2-29：2000份数据的身高和体重散点图

我们所画的直线有一个富有想象力的名字——“分类超平面”（separating hyperplane）。之所以称为“分类超平面”，是因为它把数据分成了两部分：给定一个人的身高和体重，数据点[译注3]落在超平面的一侧，你会猜测这个人是女性，而在另一侧你会猜测他是男性。这是一个相当好的猜测方法，就这份数据集而言，92%的情况下你会猜对。我们认为这个效果还不错，因为我们使用的分类模型非常简单——它仅仅用了身高和体重两个特征而已。在实际的分类任务中，我们常常要用几十个、几百个甚至几千个特征来预测类别。只不过这份数据刚好特别容易上手，我们才选了它来演示。

译注3：此处的数据点指的是由身高和体重确定的点。

本章即将结束，为了让你对下一章充满期待，我们把实现刚才预测的R代码展示出来。根据观察，根本不用什么代码，结果就出人意料：

```
heights.weights <- transform(heights.weights, Male = ifelse(
Gender == 'Male', 1, 0))
logit.model <- glm(Male ~ Height + Weight, data = heights.weights,
family = binomial(link = 'logit'))
ggplot(heights.weights, aes(x =Height, y = Weight, color = Gender)) +
geom_point() +
stat_abline(intercept = -coef(logit.model)[1]/coef(logit.model)[2],
slope = -coef(logit.model)[3] / coef(logit.model)[2],
geom = 'abline',color = 'black')
```

在第3章中，我们会倾囊相授如何用现成的机器学习方法来构建自己的分类器。

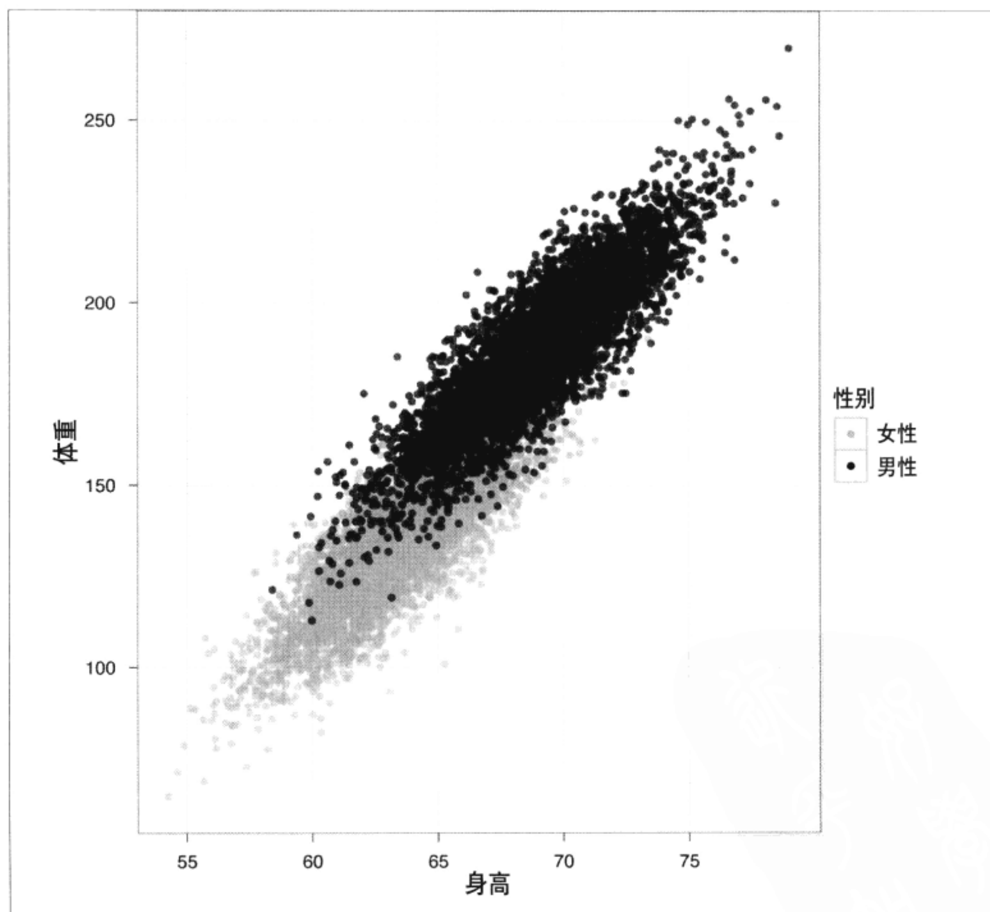


图2-30：2 000份数据的身高和体重散点图（用颜色标注性别）

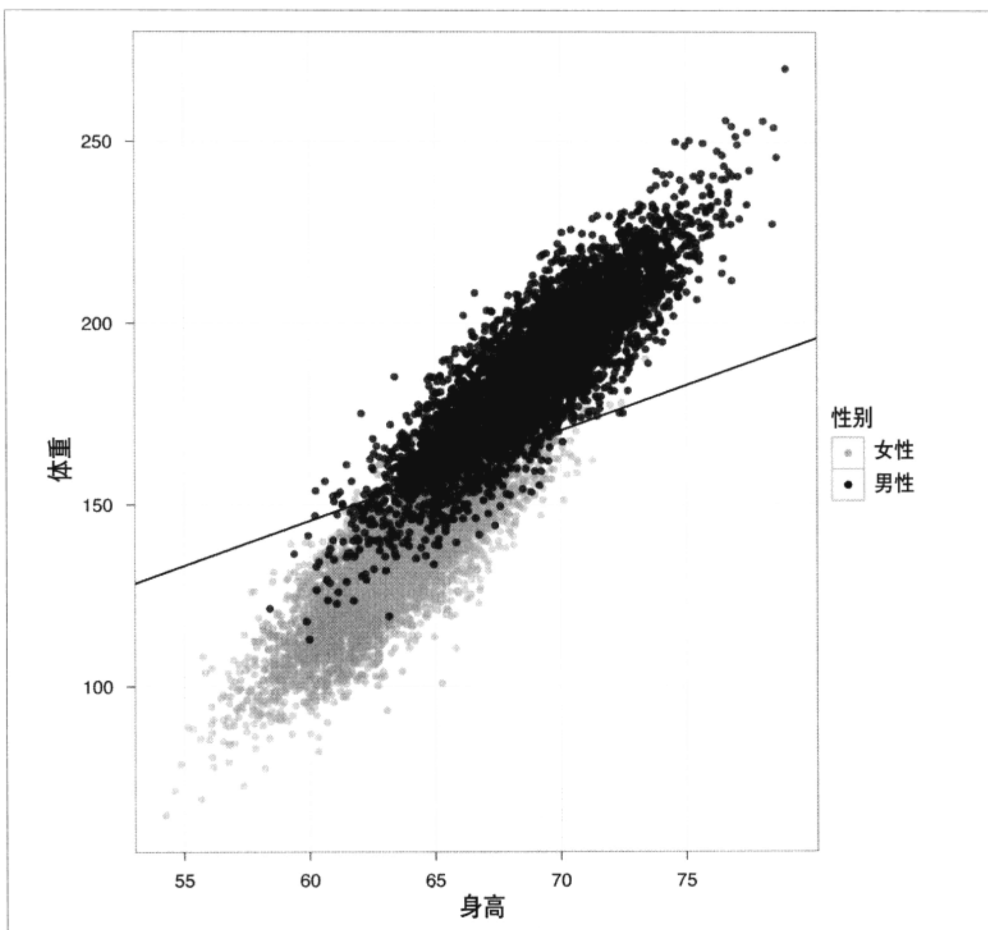


图2-31：带线性拟合的身高和体重散点图

第3章

分类：垃圾过滤

非此即彼：二分类

第2章末尾已经简要介绍过一个关于分类的案例。我们用身高和体重来预测一个人是男性还是女性。在例图中，可以用一条线把数据分成两个群体：一个为“男性”群体，另一个为“女性”群体。这条线称为分类超平面，但是从现在起，我们将使用“决策边界”（decision boundary）这个术语，因为我们有时候会遇上无法仅用一条直线来很好地完成分类的数据，比如，像图3-1所示的数据。

该图可用于描述有患病风险的人和健康的人。在图3-1中两条黑色水平线之外的区域，我们预测这些人有患病风险，但在两条水平线之间的人则健康状况良好。因此，这两条黑线便是我们的决策边界。假设图3-1中的三角形代表健康人，圆圈代表患病的人。

解决无法以一条直线作为决策边界的问题已有通用方法，这是机器学习的一大成就。本书后面内容会专门介绍一种特别的方法，即核方法（kernel trick），它处理复杂决策边界效果不错，并且几乎不增加多余的计算成本。

但是，在我们开始学习如何在实践中确定决策边界之前，要先回顾一些与分类相关的重要概念。

假定我们已有一批待学习的已标注分类样例。每个样例的构成包括：一个标签，也称为一个类别或类型；一系列用于描述样例的可测量变量。我们把这些可测量的变量叫做特征或预测变量（predictor）。比如，此前处理的身高和体重数据就是我们用于猜测“男性”和“女性”标签的特征。

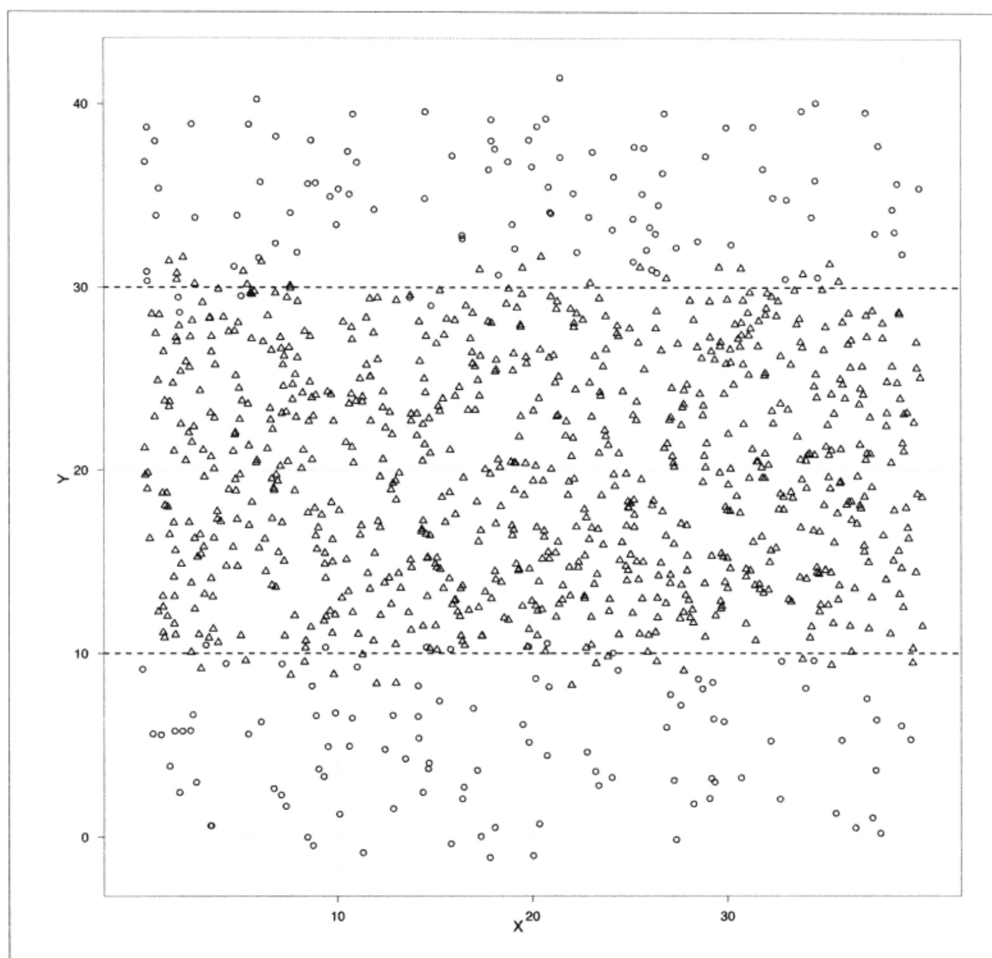


图3-1：用多个决策边界进行的分类

分类的例子比比皆是：

- 根据给定的乳房X射线图，判断病人是否患有乳腺癌？
- 从血压测量结果能否判断病人患有高血压？
- 从政治候选人的宣言能否判断他是共和党人还是民主党人？
- 判断上传到社交网络的照片中是否包含人脸？
- 《暴风雨》（The Tempest）是威廉·莎士比亚的作品还是弗朗西斯·培根的作品？

本章主要关注文本分类问题，这类问题要用到一系列方法，这些方法可以用来回答上述例子中的最后一个问题。而在案例中，要构建一个判定邮件是否是垃圾邮件的系统。

我们的原始数据来自SpamAssassin的公开语料库，可以在*http://spamassassin.apache.org/publiccorpus/*免费下载。其中一部分语料库已保存在本章的目录*code/data/*中，并且整个章节都会用到。在未处理阶段，纯文本格式的原始邮件内容就是特征。

这样的原始文本给我们提出了第一个问题。我们需要把原始文本数据转换成一组特征，从而可以用定量的方式来描述定性的概念。在本例中，具体做法就是采用0/1编码策略：垃圾或非垃圾。比如，要判定："包含HTML标签的邮件是否更可能属于垃圾邮件？"要回答这个问题，就需要把电子邮件中的文本转换成数值。幸好R中已有通用的文本挖掘程序包，可以自动完成大部分这类工作。

正因如此，人们在处理文本数据时通常都会提取一些有用的特征，而本章的重点就是培养你对这些特征的直觉。特征提取（Feature generation）是现阶段机器学习研究领域中的一个重要课题，并且还远没有达到可以用通用方法自动完成的目标。目前你最好把特征看做是机器学习中的一个基本词汇，完成的实际项目越多，你就越熟悉它。

注意： 学习一门新语言中的词汇，有助于建立一种直觉，知道什么样的才能称其为词。同理，学习人们已经用过的特征，有助于建立一种直觉，知道什么特征以后可能会用到。

根据以往经验，处理文本时所用到的最重要的特征是词频（word count）。如果我们认为HTML标签是表征电子邮件是不是垃圾邮件的重要指标，那么可以提取出诸如"html"和"table"这样的标签，分别统计它们在垃圾邮件和非垃圾邮件两类文本中出现的频次。为了展示这种方法在SpamAssassin语料库中表现如何，我们已经提前统计了"html"和"table"标签出现的频次，如表3-1所示。

表3-1："垃圾"词的频次

HTML标签	垃圾	非垃圾
html	377	9
table	1182	43

我们将数据集中的每一封电子邮件按照类型绘出了如图3-2所示的图。图3-2中并没有体现多少信息，因为太多数据点重叠了。当数据集中有一个或多个变量，而取值却大都相同时，就会经常出现这类问题。因为这个问题比较常见，所以有标准的图形化处理方法：在绘图之前，给这些值增加随机噪声。这些噪声会把数据点"分开"，从而减少重叠数量。这种添加噪声的方法称为抖动（jittering），这种方法用`ggplot2`很容易实现（见图3-3）。

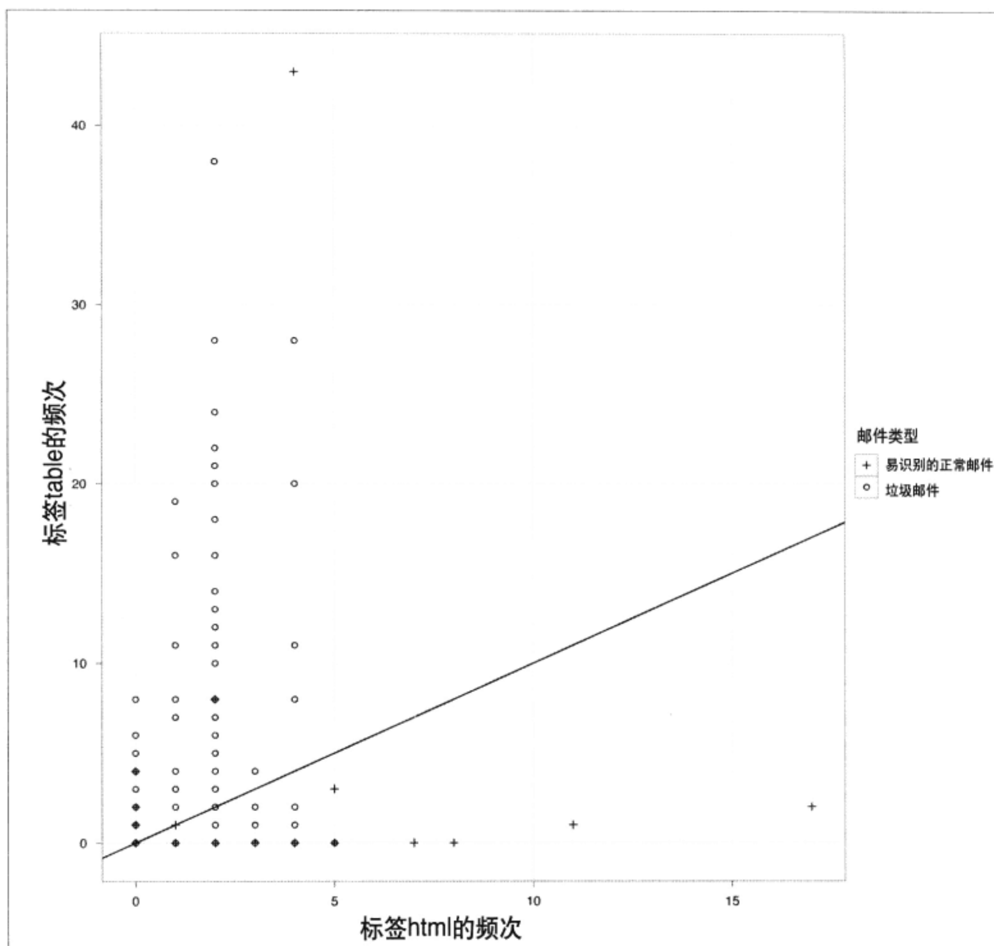


图3-2：根据邮件类型进行的“html”和“table”标签频次统计

该图说明，我们通过统计“html”和“table”标签的频次来判定邮件是否为垃圾邮件的效果其实很一般。

注意： 在本章的*code/*文件夹下的*email_classify.R*文件中，从第226行开始的命令用来生成图3-1和图3-2。

在实际操作中，除了这两个明显的特征词项之外，我们还可以使用更多的特征，从而实现更好的效果。事实上，在最终的分类器中，用到了几千个词项。尽管只用到了词频信息，但是我们仍然得到了相当准确的分类效果。在现实世界中，除了词频外，你可能想

用其他更多的特征，比如，伪造的头部信息、IP地址以及黑名单等，但目前，只介绍文本分类的基础知识。

在继续介绍文本分类之前，先回顾条件概率的一些基本概念，并讨论它们和文本分类之间的关系。

漫谈条件概率

文本分类的核心是18世纪的条件概率（conditional probability）理论在20世纪的应用。条件概率就是在已知一个事件发生的前提下，另一个事件发生的概率。比如，已知一个大学生的专业是计算机科学，我们想知道这个学生是女生的概率。这可以从调查结果中找到答案，根据美国国家科学基金会2005年的调查结果，计算机科学专业的本科生只有22%是女生[SR08]，但是，整个理工科专业的本科生有51%是女生。所以，专业为计算机科学这个条件让这个学生是女生的概率从51%下降到了22%。

本章使用的文本分类算法叫做朴素贝叶斯分类器（Naive Bayes classifier），它通过搜索文本中的两类词来判断文本类型，这两类词分别是：a）明显更可能在垃圾邮件中出现的词；b）明显更可能在非垃圾邮件中出现的词。当一个词明显更可能出现在一种情景下而非另一种时，它的出现就为判定一封新邮件是否是垃圾邮件提供了有力证据。逻辑上很简单：如果你在一封邮件中看到一个更经常出现在垃圾邮件中的词，那么它可以证明这封邮件总体来说是垃圾邮件；如果你在一封邮件中看到许多更经常出现在垃圾邮件中的词，几乎没找到经常出现在非垃圾邮件中的词，那么能够很大程度上证明这封邮件是垃圾邮件。

最终，文本分类器把上述直观感觉形式化为两个计算量：a）假设电子邮件是垃圾邮件，看到其具体内容的概率；b）假设电子邮件不是垃圾邮件，看到同样内容的概率。如果一封邮件被当做垃圾邮件，我们更可能看到其内容，那么可以断定它是垃圾邮件。

一封邮件，需要多高的可能性才值得贴上垃圾邮件的标签？这取决于另一类信息：一封垃圾邮件出现的基本概率。这个基本概率信息就是通常所说的先验概率（prior）。可以这样理解先验概率：如果你在公园看到的大多数鸟都是鸭子，那么当你某天清晨听见嘎嘎声，猜测那是只鸭子就相对保险。但是，如果你从来没在公园见过鸭子，那么你假定任何发出嘎嘎声的鸟都是鸭子就太过冒险了。至于电子邮件，之所以引入先验概率，是因为大多数发送的电子邮件都是垃圾邮件，这说明当判定为垃圾的证据不太确凿时，也足以判定其为垃圾邮件。

在下一节中，当我们写垃圾邮件文本分类器的时候会详细阐述以上这个逻辑。为了计算一封电子邮件的概率，假设所有词之间的频次统计相互独立。这种假设的正规说法叫做

统计独立性。如果引入了这种假设，但不肯定其正确性，那么我们的模型就是朴素的。因为我们还要用到电子邮件为垃圾邮件的基本概率信息，所以这种模型也叫做贝叶斯模型——这主要是为了纪念首先阐释条件概率的18世纪数学家贝叶斯。综合以上两个因素，我们的模型称为朴素贝叶斯分类器。

试写第一个贝叶斯垃圾分类器

在本章前面提到过，我们要用到SpamAssassin公开语料库来训练、测试分类器。这份数据由已标注的三类电子邮件构成：垃圾邮件（spam）、易识别的正常邮件（easy ham）、不易识别的正常邮件（hard ham），你也能想到，与易识别的正常邮件相比，不易识别的正常邮件更难和垃圾邮件区分开。比如，不易识别的正常邮件的正文通常包含HTML标签。回顾一下，识别垃圾的简单方法之一就是统计此类HTML标签的数量。为了更准确地找出不易识别的正常邮件，我们需要从更多的文本特征中引入更多的信息。为了抽取这些特征需要对电子邮件文件进行文本挖掘，这也迈出了构建分类器的第一步。

这份原始电子邮件文件中的每一封都包含邮件头部和正文两部分。一封典型的易识别的正常邮件可参照例3-1。你会注意到这篇文本中有一些有用的特征。首先，邮件头部包含大量关于邮件来源的信息。实际上，受篇幅所限，我们只在例3-1里保留了一部分头部信息。尽管头部里还包含了很多有用的信息，但是我们在分类器中却不会用到头部信息。我们比较感兴趣的不是与信息传送相关的特征信息，而是如何通过邮件正文内容本身预测一封邮件的类别。这并不意味着头部或其他信息都完全没用，实际上，所有成熟的现代垃圾过滤器都用到了邮件头部中包含的信息，比如，其中是否有部分疑似伪造的信息，邮件是否来自已知的垃圾邮件发送者，头部信息是否有字段缺失。

因为我们只关心邮件正文内容，所以需要把邮件正文文本抽取出来。如果你分析过本例要用的电子邮件文件，你会发现邮件正文总是在文件中第一个空行后开始的。在例3-1中，“Hello, have you seen and discussed this article and his approach?”（你好，请问你看过并讨论过这篇文章及其所用的方法吗？）这句话就在第一个空行后直接出现。要开始构建分类器，第一步就要利用这个规律写一个R函数读取文件，并从中抽取邮件内容文本。

例3-1：典型的“易识别的正常邮件”

```
Received: from usw-sf-list1-b.sourceforge.net ([10.3.1.13]
  helo=usw-sf-list1.sourceforge.net) by usw-sf-list2.sourceforge.net with
  esmtp (Exim 3.31-VA-mm2 #1 (Debian)) id 17hsof-00042r-00; Thu,
  22 Aug 2002 07:20:05 -0700
Received: from vivi.uptime.at ([62.116.87.11] helo=mail.uptime.at) by
  usw-sf-list1.sourceforge.net with esmtp (Exim 3.31-VA-mm2 #1 (Debian)) id
  17hsoM-0000Ge-00 for <spamassassin-devel@lists.sourceforge.net>;
  Thu, 22 Aug 2002 07:19:47 -0700
Received: from [192.168.0.4] (chello062178142216.4.14.vie.surfer.at
  [62.178.142.216]) (authenticated bits=0) by mail.uptime.at (8.12.5/8.12.5)
  with ESMTP id g7MEI7Vp022036 for
  <spamassassin-devel@lists.sourceforge.net>; Thu, 22 Aug 2002 16:18:07
  +0200
From: David H=?ISO-8859-1?B?9g==?=hn <dh@uptime.at>
To: <spamassassin-devel@example.sourceforge.net>
Message-Id: <B98ABFA4.1F87%dh@uptime.at>
MIME-Version: 1.0
X-Trusted: YES
X-From-Laptop: YES
Content-Type: text/plain; charset="US-ASCII"
Content-Transfer-Encoding: 7bit
X-Mailscanner: Nothing found, baby
Subject: [SAdev] Interesting approach to Spam handling..
Sender: spamassassin-devel-admin@example.sourceforge.net
Errors-To: spamassassin-devel-admin@example.sourceforge.net
X-Beenthere: spamassassin-devel@example.sourceforge.net
X-Mailman-Version: 2.0.9-sf.net
Precedence: bulk
List-Help: <mailto:spamassassin-devel-request@example.sourceforge.net?subject=help>
List-Post: <mailto:spamassassin-devel@example.sourceforge.net>
List-Subscribe: <https://example.sourceforge.net/lists/listinfo/spamassassin-devel>,
  <mailto:spamassassin-devel-request@lists.sourceforge.net?subject=subscribe>
List-Id: SpamAssassin Developers <spamassassin-devel.example.sourceforge.net>
List-Unsubscribe: <https://example.sourceforge.net/lists/listinfo/spamassassin-devel>,
  <mailto:spamassassin-devel-request@lists.sourceforge.net?subject=unsubscribe>
List-Archive: <http://www.geocrawler.com/redir-sf.php3?list=spamassassin-devel>
X-Original-Date: Thu, 22 Aug 2002 16:19:48 +0200
Date: Thu, 22 Aug 2002 16:19:48 +0200

Hello, have you seen and discussed this article and his approach?

Thank you

http://www.paulgraham.com/spam.html
-- "Hell, there are no rules here-- we're trying to accomplish something."
-- Thomas Alva Edison


-------------------------------------------------------
This sf.net email is sponsored by: OSDN - Tired of that same old
cell phone? Get a new here for FREE!
https://www.inphonic.com/r.asp?r=sourceforge1&refcode1=vs3390
_______________________________________________
Spamassassin-devel mailing list
Spamassassin-devel@lists.sourceforge.net
https://lists.sourceforge.net/lists/listinfo/spamassassin-devel
```

注意： 用空行分隔邮件头部和正文是协议规定的。可参考RFC822[译注1]：*http://tools.ietf.org/html/rfc822*。

译注1：RFC822规定了电子邮件的标准格式。

和其他案例一样，这里要做的第一件事就是加载要用的程序包。因为是文本分类，所以要用到tm程序包，tm代表*text mining*（文本挖掘）。一旦分类器构建完毕并测试通过，就要用到ggplot2程序包来对结果进行可视化分析。还有一步很重要的初始化工作就是为所有邮件数据文件设置路径变量。前面提到，有三类邮件：易识别的正常邮件、不易识别的正常邮件以及垃圾邮件。在本例的数据文件夹下，你会注意到每一类邮件都有两套独立的文件。我们会用第一套文件来训练分类器，用第二套文件来测试分类器。

```
library(tm)
library(ggplot2)

spam.path <- "data/spam/"
spam2.path <- "data/spam_2/"
easyham.path <- "data/easy_ham/"
easyham2.path <- "data/easy_ham_2/"
hardham.path <- "data/hard_ham/"
hardham2.path <- "data/hard_ham_2/"
```

在加载完必需的程序包，设置好路径变量后，我们可以开始将两种类型邮件转换成文本语料库了，目的是构建垃圾邮件和正常邮件的特征词项类别知识库。要完成这一步，需要先写一个函数，用它打开每一个文件，找到空行，并将该空行之后的文本返回为一个字符串向量，这个向量只有一个元素，就是空行之后的所有文本拼接之后的字符串。

```
get.msg <- function(path) {
    con <- file(path, open="rt", encoding="latin1")
    text <- readLines(con)
    # The message always begins after the first full line break
    msg <- text[seq(which(text=="")[1]+1,length(text),1)]
    close(con)
    return(paste(msg, collapse="\n"))
}
```

R语言处理文件输入输出（I/O）的方式和许多其他编程语言很相似。此处所用函数接受字符串类型的文件路径，然后以rt模式打开文件，这里rt代表以文本形式读取（read as text）。同时需要注意的是，这里的编码（encoding）指定为latin1，这是因为很多邮件包含非ASCII码字符，指定编码为latin1就可以读取这些文件。readLines函数会把所打开文件链接中的每一行文本返回为字符串向量的一个元素。一旦读入了所有的文本行，就需要定位到第一个空行，然后抽取出其后的所有文本。有了字符串向量形式的邮件正文后，先关闭文件，然后用paste函数把这个向量拼接成一个单条文本元素，同时指定参数collapse为"\n"，表示由换行符来分隔各个元素。

要训练分类器，就要从所有垃圾邮件和正常邮件中得到邮件正文。方法之一就是创建一个向量保存所有邮件正文，从而使向量的每个元素就是一封邮件的内容。最直接的R实现方式就是结合我们自己实现的函数get.msg并使用apply函数完成。

```
spam.docs <- dir(spam.path)
spam.docs <- spam.docs[which(spam.docs!="cmds")]
all.spam <- sapply(spam.docs,
function(p) get.msg(paste(spam.path,p,sep="")))
```

以垃圾邮件为例，先用dir函数得到*data/spam*下所有文件名列表。在这个路径下（以及其他所有保存邮件数据的路径下）都有一个*cmds*文件，这个文件包含一个很长的UNIX基本命令列表，用于在这些目录下移动文件。因为不希望把这些*cmds*文件包含在训练数据中，所以忽略这些文件，只保留文件名中不包含*cmds*的文件。现在，spam.docs就是一个字符向量，它包含所有用于训练分类器的垃圾邮件文件名。

要得到垃圾邮件的文本向量，用到了sapply函数，它对每一个垃圾邮件文件名应用get.msg函数，从而通过函数的返回值构建一个文本向量。

注意： 请注意，我们需要给sapply函数传入一个无名函数，目的是用paste函数把文件名和适当的路径拼接起来。这在R中是很常见的做法。

执行完这些命令后，可以用head(all.spam)来检查一下结果。你会发现每个向量元素的名称与文件名一一对应。这就是sapply函数的优势之一。

下一步是利用这个邮件向量构建一个文本语料库，这要用到tm程序包中提供的函数。一旦有了语料库形式的文本，就可以通过从邮件正文中抽取特征词项来构建垃圾邮件分类器的特征集。tm程序包有一个明显的优势，那就是清洗、规整文本的繁重工作都在后台完成。用几行R代码就能完成的任务，如果自己用较低级的语言来实现，则需要很多行的字符串处理代码才能完成。

量化垃圾邮件特征词项频率的方法之一就是构造一个词项-文档矩阵（Term Document Matrix，TDM）。顾名思义，TDM是一个N × M矩阵，矩阵的行对应在特定语料库的所有文档中抽取的词项，矩阵的列对应该语料库中所有的文档。在这个矩阵中，一个[i, j]位置的元素表示词项i在文档j中出现的次数。

和之前一样，定义一个简单的函数get.tdm，该函数输入邮件的文本向量，输出TDM：

```
get.tdm <- function(doc.vec) {
    doc.corpus <- Corpus(VectorSource(doc.vec))
    control <- list(stopwords=TRUE, removePunctuation=TRUE,
        removeNumbers=TRUE,
        minDocFreq=2)
    doc.dtm <- TermDocumentMatrix(doc.corpus, control)
    return(doc.dtm)
}

spam.tdm <- get.tdm(all.spam)
```

tm程序包提供了若干方法用于构建语料库（corpus对象）。在这个案例中，因为要用邮件向量构建语料库，所以用到了VectorSource函数来构建source对象。要查看其他可用的数据源类型[译注2]，可以在R控制台上输入?getSources。通常在使用tm程序包时，一旦加载了来源文本，会将corpus函数与VectorSource函数配合使用，从而创建一个corpus对象（语料库对象）。然而，在创建一个TDM之前，我们需要告诉tm该如何清洗和规整文本。为此，要用到一个control变量，它是一个选项列表，用于设定如何提取文本。

在这个案例中用到了四个选项。首先，设定stopwords=TRUE，这个参数告诉tm在所有文本中移除488个最常见的英文停用词。要查看停用词列表，在R控制台上输入stopwords()。接下来，参数removePunctuation（移除标点符号）和removeNumbers（移除数字）都设定为TRUE，原因当然再明白不过，主要是为了减少与这些字符有关的噪声——尤其因为许多文档都包含HTML标签。最后，设定minDocFreq=2，这确保只有那些在文本中出现次数大于1次的词才能最终出现在TDM的行中。

现在，我们已经基本处理完毕垃圾邮件，可以开始构建分类器了。具体来说，首先可以用TDM来构建一套垃圾邮件的训练数据。要在R中实现这一目标，推荐方法是构建一个数据框来保存所有特征词项在垃圾邮件中的条件概率。正如前面的计算机科学专业女生那个例子所做的一样，需要训练分类器，使之能在已知观测特征的前提下计算出邮件是垃圾的概率。

```
spam.matrix <- as.matrix(spam.tdm)
spam.counts <- rowSums(spam.matrix)
spam.df <- data.frame(cbind(names(spam.counts),
     as.numeric(spam.counts)),stringsAsFactors=FALSE)
names(spam.df) <- c("term","frequency")
spam.df$frequency <- as.numeric(spam.df$frequency)

spam.occurrence <- sapply(1:nrow(spam.matrix),
     function(i) {length(which(spam.matrix[i,] > 0))/ncol(spam.matrix)})
spam.density <- spam.df$frequency/sum(spam.df$frequency)

spam.df <- transform(spam.df, density=spam.density,
     occurrence=spam.occurrence)
```

为了构建这个数据框，首先要用as.matrix命令把TDM对象转换成R的标准矩阵。然后用rowSums命令创建一个向量，该向量包含每个特征在所有文档中的总频次。因为要用data.frame函数把一个字符向量和一个数值向量结合在一起，所以默认情形下R会把这些向量转换成常见表达形式。频次可能是字符形式，因此它会转换成因子类型，于是就要注意设定stringsAsFactors=FALSE。接下来，需要处理一些琐事，包括修改列名，把频次数转换成数值向量。

译注2：数据源类型是指构造corpus对象的参数类型source。

通过接下来的两个步骤，生成关键的训练数据。第一步，计算一个特定特征词项所出现的文档在所有文档中所占的比例。通过sapply把每一行的行号传入一个无名函数，该函数统计该行中值为正数的元素个数，然后除以TDM中列的总数——也就是除以垃圾邮件语料库中的文档总数。第二步，统计整个语料库中每个词项的频次（我们并不会使用这些频次信息来分类，但是如果想知道某些词是否影响结果，对比频次数相当有用。）

最后，用transform函数把向量spam.occurrence和spam.density加入数据框中。现在，分类器的训练数据已经准备完毕！

现在，我们来检查一下，看看这份训练数据中哪些特征词项在垃圾邮件中指示性最明显。为此，我们按照occurrence列对spam.df进行排序，并看看前几条数据：

```
head(spam.df[with(spam.df, order(-occurrence)),])
        term frequency         density  occurrence
2122    html       377 0.005665595       0.338
538     body       324 0.004869105       0.298
4313    table     1182 0.017763217       0.284
1435    email      661 0.009933576       0.262
1736    font       867 0.013029365       0.262
1942    head       254 0.003817138       0.246
```

我们之前反复提到，HTML标签好像是垃圾邮件中最明显的文本特征。在训练数据中，超过30%的垃圾邮件包含标签html，以及其他常见的HTML相关标签，比如body、table、font以及head。请注意，这些HTML标签的频次统计（frequency列）并非最高。你可以自行把上面代码中的-occurrence替换为-frequency就知道结果了。从定义分类器的角度来说，这非常重要。如果采用frequency（总词频）和紧随其后的density（词的概率）作为训练数据，就会把某些类型的垃圾邮件权重调得过高——尤其是包含table标签的那些垃圾邮件。然而，我们知道并非所有的垃圾邮件正文都是用这种方式生成的。综上所述，较好的方法是：根据有多少邮件包含这个特征词项来定义一封邮件是垃圾邮件的条件概率。

既然已有了垃圾邮件的训练数据，就需要正常邮件的训练数据，从而让总体样本平衡。作为练习的一部分，只用易识别的正常邮件数据来构建训练数据。当然，也可以在其中包含不易识别的正常邮件，事实上，如果是为了打造一个真实的垃圾邮件自动识别系统，训练数据就应该包含那些不易识别的正常邮件。但就这个案例来说，仅用一批容易分类的语料库文档来训练有助于观察文本分类器的工作原理。

构造正常邮件的训练数据和此前构建垃圾邮件训练数据的方法一模一样，所以就不再重复书写那些命令了。和构造垃圾邮件训练数据唯一不同之处就是，只用data/easy_ham文件夹里的前500封邮件数据。

你可能已经注意到，在这个文件夹下有2500封正常邮件。那么为什么我们要忽略五分之四的数据呢？在构建第一个分类器的过程中，假定每一封邮件是垃圾邮件或正常邮件的概率是相等的。因此，最好保证数据能反映假设，我们只有500封垃圾邮件数据，因此也把正常邮件限定在500封。

注意： 如果想知道如何把正常邮件数量限定在500封，请查看本章代码文件*email_classify.R*中第168行。

构造好正常邮件的训练数据之后，我们可以像查看垃圾邮件训练数据一样进行查看，从而可以对比这两份数据：

```
head(easyham.df[with(easyham.df, order(-occurrence)),])
        term  frequency      density  occurrence
3553   yahoo        185  0.008712853       0.180
966     dont        141  0.006640607       0.090
2343  people        183  0.008618660       0.086
1871   linux        159  0.007488344       0.084
1876    list        103  0.004850940       0.078
3240    time         91  0.004285782       0.064
```

我们首先注意到，在正常邮件训练数据中，特征词项分布更稀疏。在文档中出现频率最高的词“yahoo”也只在18%的文档中出现过，其他词都只在小于10%的文档中出现过，但是，垃圾邮件训练数据前几个词项均在大于24%的文档中出现过。我们可以想想，如何利用这种差异把垃圾邮件和正常邮件区分开。如果一封邮件仅含有一两个与垃圾邮件非常相关的词，就需要很多非垃圾词汇才能把它分类为正常邮件。定义两类训练集之后，接下来准备完成分类器并开始测试它！

定义分类器并用不易识别的正常邮件进行测试

我们要定义的分类器是以一封邮件正文为输入，然后计算其是垃圾邮件或正常邮件的概率。幸运的是，我们已经实现了大多数的函数并且已经生成了计算要用的数据。但是在开始之前，还有一个关键的难点必须考虑。

我们需要决定如何处理新邮件中那些能在训练集中找到的特征词项，以及那些在训练集中找不到的词项（见图3-3）。为了计算一封邮件是垃圾邮件或正常邮件的概率，需要找出待分类邮件和训练集共有的词项。然后，用这些特征的概率计算这封邮件是训练集中对应类别的条件概率，这很直接，但是当待分类邮件中的词项未在训练集中出现时，应该怎么办呢？

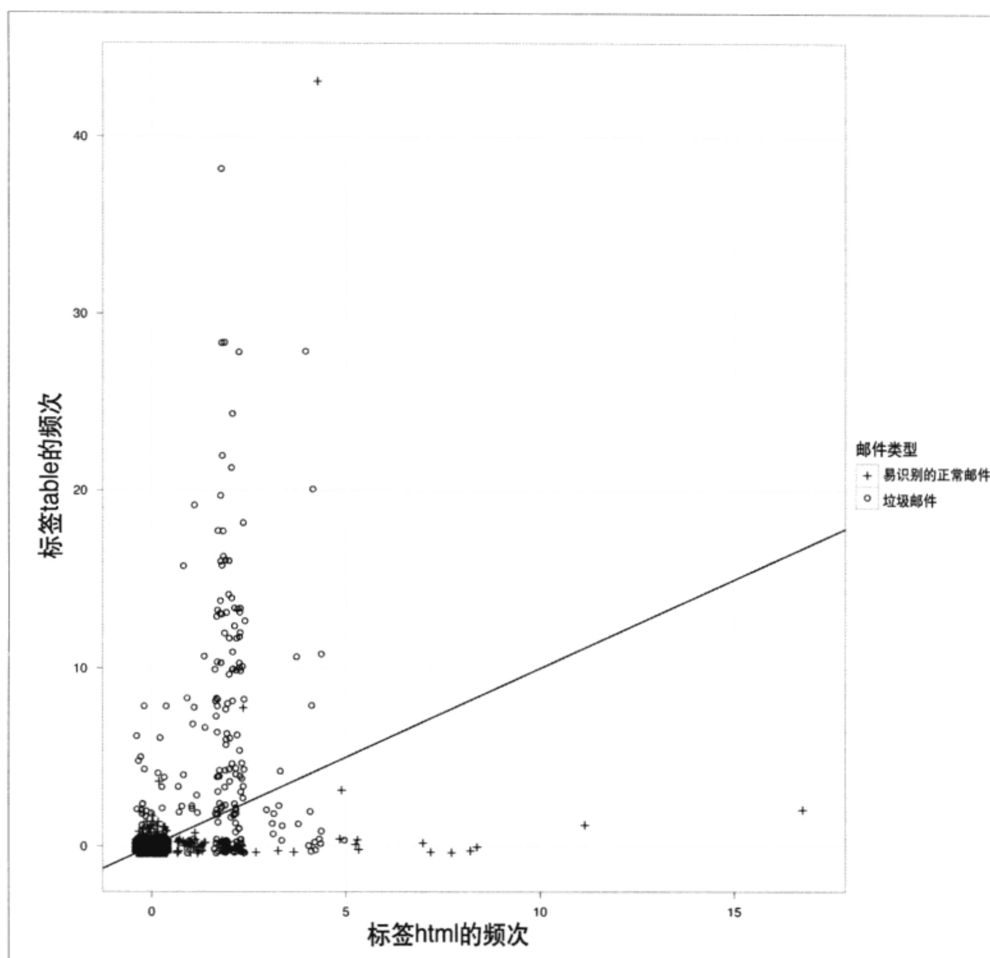


图3-3：按邮件类型绘制的特征html和table抖动频率图

要计算一封邮件的条件概率，我们的方式是计算各个特征词项在训练集中的概率乘积。比如，HTML标签html在垃圾邮件中出现的概率是0.30，HTML标签table在垃圾邮件中出现的概率是0.10，那么可以说在垃圾邮件中同时看到这两个词项的概率是$0.30 \times 0.10 = 0.03$。但是对于那些没在训练集中出现的词项来说，我们对它们在垃圾邮件和正常邮件中出现的概率一无所知。有一种可能的办法是假设它们在这些类别中出现的概率是0，因为我们没有在训练集中观测到。然而，这种方法大错特错。首先，就因为没有观测到，我们就断定它们永远不会在所有邮件中出现，这也太无知了。再者，条件概率是通过乘积来计算的，假如我们把没在训练集中出现过的词项概率赋值为0，学过小学数学都知道，大多数邮件的条件概率都会是0，因为一遇到未知的词项时都会把所有概率值

乘以0。这对我们的分类器而言，会造成灾难性的后果，因为很多甚至所有邮件作为垃圾或正常邮件的概率都会被错误地计算为0。

研究人员已经提出了许多很妙的办法来应付这个问题，比如依据某些分布给它们赋一个随机概率，或者用自然语言处理（Natural Language Processing，NLP）技术估计一个词项在特定上下文中的“垃圾倾向”。我们的方法是简单地给这些没在训练集中出现的词项赋予一个特别小的概率。实际上这也是在简单文本分类器中处理缺失特征的常见做法，而且也能很好地满足我们的要求。在这个案例中，我们把这个概率默认设置为0.0001%，这对这份数据集来说已经足够小了。最后，因为假设每一封邮件是垃圾邮件和正常邮件的可能性相同，所以把每一个类别的先验概率都默认设置为50%。然而因为后面还会涉及这个问题上来，所以我们实现了一个classify.email函数，如此一来，这个先验概率就可变了。

警告： 对于未出现在训练集中的特征，我们所给的概率值0.0001%也不是放之四海而皆准的。虽然适用于这个案例，但是在其他情况下它可能太大或者太小，那么你所构建的系统可能就完全没用了。

```
classify.email <- function(path, training.df, prior=0.5, c=1e-6) {
    msg <- get.msg(path)
    msg.tdm <- get.tdm(msg)
    msg.freq <- rowSums(as.matrix(msg.tdm))
    # Find intersections of words
    msg.match <- intersect(names(msg.freq), training.df$term)
    if(length(msg.match) < 1) {
        return(prior*c^(length(msg.freq)))
    }
    else {
        match.probs<-training.df$occurrence[match(msg.match, training.df$term)]
        return(prior*prod(match.probs)*c^(length(msg.freq)-length(msg.match)))
    }
}
```

你会注意到，在classify.email函数中前三步和我们在训练阶段所做的事一样。我们也得用get.msg函数抽取出邮件正文，并用get.tdm将其转换成TDM，最后用rowSums计算特征词项的频率。接下来，我们要找到出现在邮件中的词项和出现在训练集中的词项的交集，如图3-4所示。为此，我们用到intersect（交集）命令，给它输入邮件中找到的特征和训练集中的词项。返回的数据如图3-4中深灰色区域所示。

分类的最后一步是判断邮件中是否有词项出现在训练集中，如果有，那就用这些特征词项来计算该邮件属于训练集对应类别邮件的概率。

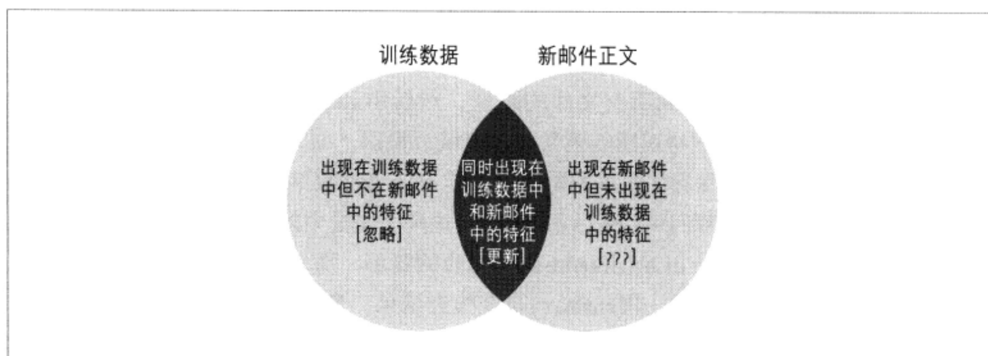


图3-4：新邮件的处理策略

假设现在我们就要开始判定一封邮件是否为垃圾邮件。msg.match将保存这封邮件中的所有在训练集spam.df中出现过的特征词项。如果交集为空，那么msg.match的长度就比1小，于是就用先验概率乘以小概率值c的邮件特征数次幂（prior*c^(length(msg.freq))）。得到的结果就是这封邮件被分类为垃圾邮件的概率，值很小。

相反，如果这个交集不为空，我们需要找出这些同时出现在训练集和新邮件中的特征词项，然后查出它们在文档中出现的概率（occurrence）。使用match函数完成查找，它能找到词项在训练数据的term列中出现的位置。我们根据这些位置可以从occurrence列中返回特征所对应的文档概率，返回的值保存在match.probs中。然后，计算这些返回值的乘积，并将乘积结果再与下列值相乘：邮件为垃圾邮件的先验概率、特征词项的出现概率以及缺失词项（未出现在训练集中的词项）的小概率。获得的结果就是在已知邮件中有哪些词项出现在训练集中后，对于它是垃圾邮件的贝叶斯概率估计值。

作为初步的测试，我们用垃圾邮件和易识别的正常邮件来训练，再对不易识别的正常邮件进行分类。因为这些邮件都是正常的，所以理想情况下分类器会给它们作为正常邮件赋予一个较高的概率值。然而我们知道，那些不易识别的正常邮件很难进行分类，因为它们包含的很多特征也同样存在于垃圾邮件中。那么现在就让我们一睹这个简易分类器的表现吧：

```
hardham.docs <- dir(hardham.path)
hardham.docs <- hardham.docs[which(hardham.docs != "cmds")]

hardham.spamtest <- sapply(hardham.docs,
    function(p) classify.email(file.path(hardham.path, p),
    training.df = spam.df))

hardham.hamtest <- sapply(hardham.docs,
    function(p) classify.email(file.path(hardham.path, p),
    training.df = easyham.df))
```

```
hardham.res <- ifelse(hardham.spamtest > hardham.hamtest, TRUE, FALSE)
summary(hardham.res)
```

我们像以前一样需要依次得到所有文件的路径，然后用sapply封装了对垃圾邮件和正常邮件的测试，最后再用不易识别的正常邮件测试分类器。向量hardham.spamtest和hardham.hamtest中分别保存了每一封邮件在给定对应训练数据的前提下是垃圾或正常邮件的条件概率计算结果。我们用ifelse命令比较这两个向量中的概率值，如果hardham.spamtest中的概率大于hardham.hamtest中对应的概率，那么分类器就判定其为垃圾邮件，反之就是正常邮件。最后，用summary命令检查结果，参见表3-2。

表3-2：用不易识别的正常邮件来测试分类器

邮件类型	分类为正常邮件的数量	分类为垃圾邮件的数量
不易识别的正常邮件	184	65

恭喜！你已经写好了第一个分类器，而且它可以较出色地将不易识别的正常邮件识别为非垃圾邮件。在这个案例中，误判率（误判为垃圾邮件的比例）约为26%。也就是说，有四分之一的不易识别的正常邮件被误判为垃圾邮件了。你可能觉得这个效果不够好，在实际中我们也不愿看到邮件平台有这样的结果，但是考虑到这个分类器只是一个简易版，它的表现还是不错的。当然，更好的测试不仅仅只看分类器对不易识别的正常邮件的表现，还要看其对易识别的正常邮件和垃圾邮件的表现如何。

用所有邮件类型测试分类器

第一步就是实现一个简单的函数，用于一次性对所有邮件完成像上一节那样的概率比较。

```
spam.classifier <- function(path) {
    pr.spam <- classify.email(path, spam.df)
    pr.ham <- classify.email(path, easyham.df)
    return(c(pr.spam, pr.ham, ifelse(pr.spam > pr.ham, 1, 0)))
}
```

为了简单起见，spam.classifier函数将根据训练数据spam.df和easyham.df判定邮件是否为垃圾邮件。如果邮件是垃圾邮件的概率大于它是正常邮件的概率，函数就返回1，否则返回0。

本案例的最后一步，我们要用简易分类器测试垃圾邮件、易识别的正常邮件和不易识别的正常邮件的第二套数据。这些过程和上一节所做的基本相同：把spam.classifier函数封装在lapply函数中，传入文件路径，生成一个数据框。因此，此处就不再列出这些函数了，但是鼓励你参考文件email_classifier.R中自第259行开始的内容，看看具体如何完成这些测试。

这个新的数据框包含三个数据集中每封邮件作为垃圾邮件或正常邮件的概率、分类结果、邮件标注类型。这个新的数据集叫做class.df，用head命令可以查看其内容：

```
head(class.df)
         Pr.SPAM          Pr.HAM   Class    Type
1  2.712076e-307   1.248948e-282   FALSE    EASYHAM
2    9.463296e-84    1.492094e-58  FALSE    EASYHAM
3    1.276065e-59    3.264752e-36  FALSE    EASYHAM
4    0.000000e+00  3.539486e-319   FALSE    EASYHAM
5    2.342400e-26    3.294720e-17  FALSE    EASYHAM
6  2.968972e-314   1.858238e-260   FALSE    EASYHAM
```

从前六条结果可以看出似乎分类器的表现还不错，但是，我们来计算一下分类器在整个数据集上的误判率和漏判率（漏判为垃圾邮件的比例）。为此，我们需要用分类结果构造一个N × M矩阵，行是实际的邮件类型，列是预测的邮件类型。因为把三类邮件分成两类，所以我们的混淆矩阵有三行两列（见表3-3）。每一列就是预测为正常邮件或者垃圾邮件的百分比，如果我们的分类器表现完美，那么分类结果里的两列相应的值应该是[1,1,0]和[0,0,1]。

表3-3：分类结果矩阵

邮件类型	分类为正常邮件的比例（%）	分类为垃圾邮件的比例（%）
易识别正常邮件	0.78	0.22
不易识别正常邮件	0.73	0.27
垃圾邮件	0.15	0.85

虽然分类器效果还是相当不错，不过，并不完美。和第一步测试一样，仍然有约25%的误判率，其中分类器对易识别正常邮件的分类效果略好于对不易识别的正常邮件。另一方面，垃圾邮件的漏判率也更低，仅15%。为了对分类器的预测结果有更深的认识，我们用散点图把结果绘制出来，其中x轴是邮件作为正常邮件的概率，y轴是作为垃圾邮件的概率。

图3-5展示了log-log刻度绘制的散点图。用对数（log）转换的原因是许多预测概率非常小，而另一些又不是那么小。因为差别悬殊，所以不太容易直接比较结果。要把可视化刻度转换成更易比较的值，取对数是一种简便方法。

我们也在图中添加了一个简单的决策边界，就是y=x，或者说是一个完美的线性关系。之所以有这样的情况，主要是因为这个分类器要比较邮件是垃圾邮件或正常邮件的预测概率，然后基于哪个概率更大再决定其类别。所以黑色决策线之上的点都应该代表垃圾邮件，之下的点都应该代表正常邮件。但你可以看到，事实并非如此，而是邮件类型有较大程度的混淆。

至于在误判率上分类器表现不佳的原因，从图3 5上也能得到一个直观的感觉。看上去，分类失败有两个普遍的可能原因。第一，许多不易识别的正常邮件被判定为垃圾邮件的概率大于0，而它被判定为正常邮件的概率却几乎为0，在图中就是那些落在y轴上的点。第二，易识别的正常邮件和不易识别的正常邮件被判定为正常邮件的相对概率都很高[译注3]。

观察到这两个现象都说明不易识别的正常邮件的训练数据不足，因为显然还有很多与正常邮件相关的特征现在还没有被纳入训练数据。

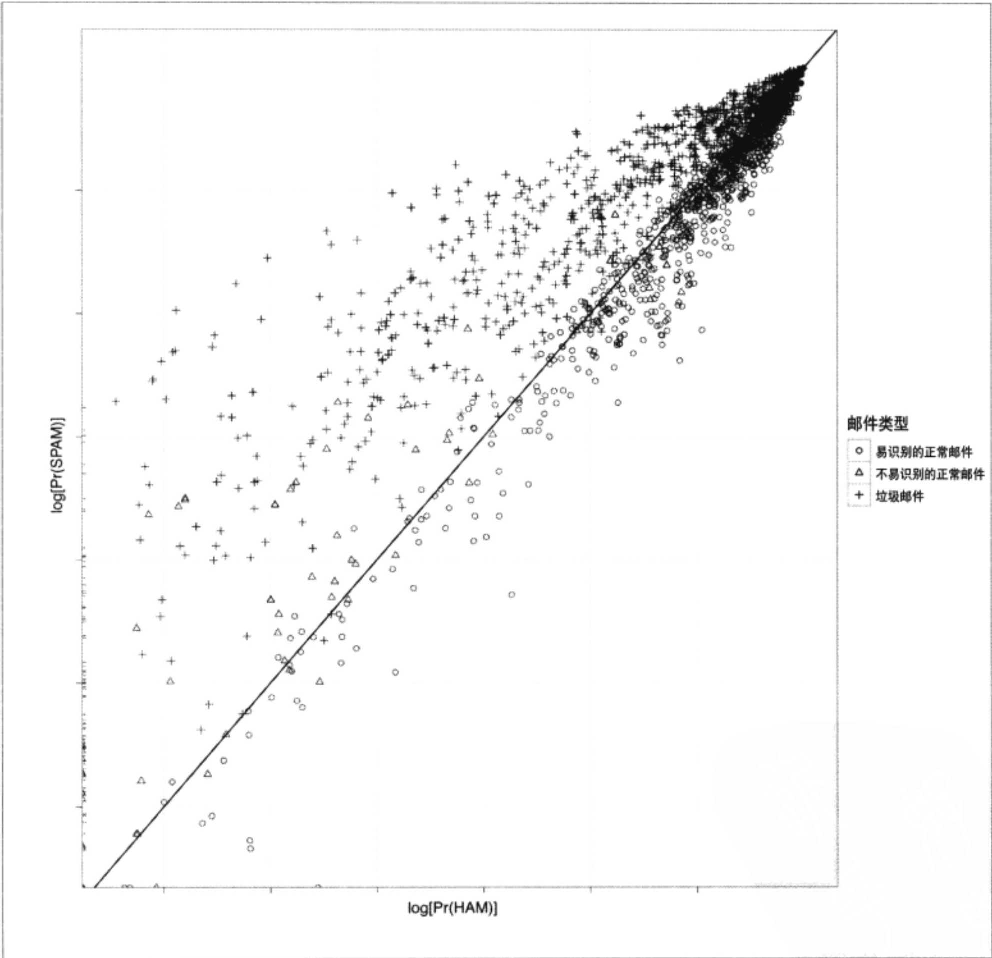


图3-5：邮件分类的预测概率散点图，log-log刻度

译注3：这说明样本并未从概率上体现出二者的区分度。

效果改进

本章介绍了文本分类的概念。为此，我们用最少的假设和特征构造了一个简易的贝叶斯分类器。这种分类器的核心是经典的条件概率理论在当代的应用。尽管我们只用到了现有整个数据的一小部分来训练分类器，但是，这种简单的方法却表现得可圈可点。

我们也提到了，误判率和漏判率远远达不到垃圾过滤器的实用水准。也正如整章过程中我们一贯强调的那样，有许多简单的调整方法可用来提高现有模型的效果。比如，我们的方法是假设邮件作为垃圾邮件和正常邮件的先验概率相同。然而实际上，我们知道正常邮件和垃圾邮件的比例关系接近80%比20%。那么，改善效果的方法之一就是修改先验概率来反映这个事实，并重新计算预测概率：

```
spam.classifier<-function(path) {
    pr.spam<-classify.email(path, spam.df, prior=0.2)
    pr.ham<-classify.email(path, easyham.df, prior=0.8)
    return(c(pr.spam, pr.ham, ifelse(pr.spam > pr.ham, 1, 0)))
}
```

你应该还记得我们在`classify.email`函数中留下了一个`prior`参数可以调节，因此只需将前面案例的`spam.classify`做这样一个小改动即可。我们要重新运行分类器，并对照结果，而且鼓励你也这样做。然而，这个新的假设与我们数据中两类邮件的分布并不相符。为了更加准确，我们应该用整个易识别正常邮件的数据集重新训练分类器。你应该还记得我们之前只用了原始正常邮件数据的前500封，目的是使训练数据与贝叶斯假设一致。按照这个逻辑，为了与新的假设一致，我们必须引入整个数据集。

当用新的`easyham.df`和`classify.email`参数更新了分类器后，我们看到，在误判率方面，分类器的性能明显得到了提升（见表3-4）。

表3-4：改善后的分类器结果矩阵

邮件类型	分类为正常邮件的比例（%）	分类为垃圾邮件的比例（%）
易识别的正常邮件	0.90	0.10
不易识别的正常邮件	0.82	0.18
垃圾邮件	0.18	0.82

就是这个小小的改变，误判率降低了50%！然而有趣的是，通过这一方法改善性能后，漏判率却惨不忍睹。本质上讲，我们所做的其实是移动了决策边界（回顾图3-1）。这种方式是用漏判率的上升换来了误判率的下降。这个案例很好地说明了为什么模型指定很关键，每个假设和特征的选择是如何影响结果的。

下一章，我们要拓宽视野，不再只关注简单的二分类——非此即彼——这样的案例。本章一开始就提到，通常很难或几乎不可能基于单个决策边界对观测记录进行分类。下一章我们要探索如何基于与高优先级有关的特征对电子邮件进行排序。尽管分类任务范围不断拓宽，模型的特征选择依然很重要。下一章你会看到数据如何限制了特征的选择，以及这对模型设计的影响有多严重。

第4章

排序：智能收件箱

次序未知时该如何排序

第3章中我们详细讨论了二分类的概念——也就是把对象判定为两个类别中的其中一个。在很多情况下，能够做出这样的辨别，我们就满意了。但是，如果同一个类别中每个对象并非被同等创建，并且我们想在这个类别中对它们进行排序那又该当如何？举个简单的例子，假如我们想说某一封邮件垃圾倾向最高，另一封的垃圾倾向程度次之，或者用其他有效的方式去对它们进行区分，那该怎么办呢？设想一下，我们不仅要过滤垃圾邮件，还要将更重要的邮件置顶在收件箱列表中。这个问题在机器学习中很常见，也将是本章的重点。

通过产生一组规则对一个对象列表排序，这在机器学习中越来越常见，只不过你可能还未从专业角度思考过。你更可能听说过类似推荐系统的东西，它就是在后台对产品进行了排序。即便你可能连推荐系统也没听说过，但是你肯定在某些场合使用过或者与之交互过。一些非常成功的电子商务网站已经从中尝到甜头了，他们利用其用户数据为用户推荐可能感兴趣的其他产品。

举个例子，如果你曾经在亚马逊（Amazon.com）上买过东西，那么你就与推荐系统交互过。亚马逊要解决的问题很简单：你最可能购买他们列表上的哪些库存商品？这种说法的隐含意义就是：亚马逊列表上的库存商品对每个用户来说都有一个特定的顺序。同样，在Netflix.com上有海量的DVD可以出租给用户。为了让用户能在这个网站上找到尽可能多感兴趣的DVD，Netflix部署了一套复杂的推荐系统用于为人们提供租赁建议。

这两家公司的推荐系统都基于两种数据。第一种是库存商品本身的数据。对亚马逊来

说，如果商品是一台电视机，那么这种数据可能包含其类型（比如：等离子、LCD、LED）、制造商、价格等。对Netflix来说，这个数据也许是电影的体裁、演员阵容、导演、片长等。第二种是消费者浏览和购买行为数据。这类数据可以帮助亚马逊了解大多数用户在购买一台新等离子电视机时需要什么配件；可以帮助Netflix了解George A. Romero[译注1]的粉丝们最经常租赁的是哪些浪漫喜剧。这两种数据的特征很容易确定，我们知道什么样的数据是分类数据标签，比如产品类型或电影体裁，而且用户产生的数据以购买/租赁记录以及评分等形式已经被很好地结构化。

在进行排序时，通常已有明确的输出实例，这种机器学习问题一般称为有监督学习（supervised Learning）。与之相对的是无监督学习（unsupervised learning），无监督学习在开始处理数据时预先并没有已知的输出实例。要对两者的区别理解得更透彻，可以把有监督学习看成是经过指导的学习过程。比如，你要教一个人烤樱桃派，你可能会给他一份食谱，然后让他尝尝自己做出来的派。尝过之后，他可能会根据味道对配料再做一些调整。有了一份用过的配料表（也就是输入），也有了做出来的味道（也就是输出），这意味着他可以分析每一种配料有多大的作用，从而找到一份制作美味樱桃派的完美食谱。

相反，如果你只知道炸豆泥可以搭配炸玉米粉圆饼，而烤樱桃可以和面团搭配，那么你也许能够把这些调料划分成不同的类型[译注2]，有些可以用来做墨西哥菜，有些可以用来做美式甜点。实际上最常见的无监督学习就是聚类，聚类就是把对象按照其相同点或不同点分别归入一定数目的组内。

如果你已经读过第3章并且做过其中的练习，那么你已经解决了一个有监督学习的问题了。在垃圾邮件分类中，我们已经知道了与垃圾邮件和正常邮件内容相关的词项，并依据这个指导训练了一个分类器。因为那里的问题很简单，所以虽然只用了一种特征集：邮件内容的词项，但是得到的分类效果还不错。对于排序来说，需要给每个特征词赋予一个唯一的权重，以此来更好地对其进行分层并排定次序。

因此，接下来的内容中将专门回答本节题目中的问题：次序未知时该如何排序？你可能也猜到了，具体到根据邮件重要性进行排序，需要将问题变成：在邮件数据中如何得到特征，以及这些特征如何与邮件的优先级挂钩？

按优先级给邮件排序

哪些因素决定了邮件的重要性？要回答这个问题，先退一步想想什么是邮件。首先，它

译注1：George A. Romero是美国电影导演。

译注2：因为没有人教你如何划分，所以称为“无监督”。

是一种基于事务的媒介。人们在不断地收发消息。因此，要确定邮件的重要性，需要关注事务本身。在垃圾分类中，可以利用所有邮件的静态信息来决定其类型，而与之不同的是，要根据重要性给邮件排序，必须关注往来事务本身的动态信息。具体地讲，我们要做的就是确定一个人在收到一封新邮件后马上处理的可能性有多大。换句话说，假设已经选择好特征集，那么收件人会在短时间内处理这封邮件的可能性有多大。

这个问题涉及的一个新的关键维度就是时间。在一个基于事务的情境中，要对事件的重要性排序就得考虑时间因素。用时间来决定邮件的重要性，很自然的方法就是计算收件人在收到邮件后过了多少时间才处理这封邮件。在给定特征集下，这个平均时间越短，那么这封邮件在所属类型中的重要性就越高。

这个模型的隐含假设就是越重要的邮件越早被处理。直观上感觉这是讲得通的。每个人在盯着收件箱浏览邮件时，都可以将需要立即处理的和可以稍后处理的邮件分辨得清清楚楚。我们这样自然而然做出的区分正是下面几节要让算法实现的功能。但是在开始之前，必须确定邮件内容里的哪些特征可以很好地表征邮件优先级。

邮件优先级的特征

如果你用过Google的Gmail邮箱服务，就会知道Google在2010年最先开始普及“智能收件箱”这一概念。当然，正是因为这个给了我们灵感在本章探讨排序案例研究，所以在设计自己的排序算法之前，有必要先复习一下Google所实现的排序算法。Google在公布智能收件箱的特征几个月之后，他们发表了一篇题为“The Learning Behind Gmail Priority Inbox”的论文，文中描述了他们的有监督学习的设计策略，以及怎么成规模地实现这个策略[DA10]。出于本章的目的，我们只对前半部分感兴趣，但是作为本章的补充读物，强烈推荐你完整阅读这篇论文，而且一共才四页，这点时间值得花。

前面提到过，对时间的测量最关键，而在Google的项目里，他们拥有长时间范围内用户与邮件的详细交互记录。具体地说，Google的智能收件箱想预测的是用户在收到邮件后的几秒内对邮件采取行为的概率。用户在Gmail里面可以采取的行为种类很多：阅读、回复、标记等。这里所说的收到邮件（delivery）的时间不完全等同于服务器收到邮件的时间，而是用户收到邮件的时间，即当用户查看邮箱时。

和垃圾邮件分类相比，这是一个相当好表述的问题：我们首先定义一些可能的行为集合（阅读、回复、标记等），然后定义一个时间区间（比如，用户看到邮件后1~10秒），我们需要判断，在定义的时间区间内，用户会执行定义的行为集合中某些行为的概率是多少？

在海量的邮件特征中，Google决定把重点放在哪些特征上？你可能也想到了，他们采用

的特征数量非常大。正如该论文作者所述，和垃圾分类不同——垃圾分类对所用用户几乎没有差别——每个人对邮件的优先级都有不同的排序方式。鉴于用户在评价特征集这方面的差异性，Google的方法需要采用多个特征。为了开始设计算法，Google的工程师探索了邮件各种各样的特征，他们对此是这样描述的：

> 有几百个特征，可以分为几大类别。社交特征（social feature）基于收件人和发件人之间的交互程度，比如某个发件人的邮件被收件人阅读过的百分比。内容特征（content feature）用于识别和收件人对邮件采取行为与否高度相关的最近特征词和头部信息，比如，一个最近特征词在主题中的出现。最近用户的特征词是在学习之前预处理阶段发现的。线程特征（thread feature）记录用户在当前线程下的交互行为，比如，用户开始一个新线程。标签特征（label feature）检查用户通过过滤器给邮件赋予的标签。我们在排序过程中计算特征值，并临时存储特征值以备之后学习使用。通过对连续型特征值的直方图使用类似ID3的简单算法，可以将其自动地划分为二值特征。

前面已经提到过，Google拥有的用户与Gmail的交互记录时间跨度非常长，因此，关于用户何时对邮件采取何种行为，他们有很全面的认识。遗憾的是，在我们的案例中，拿不到这样详细的用户日志。于是，我们还是用SpamAssassin公开语料库来代替，这个语料库在*http://spamassassin.apache.org/publiccorpus/*上可以免费下载。

尽管发布这个数据集的初衷是用来测试垃圾分类算法的，但是其中也包含了每一个用户邮件的时间线（timeline），这很方便。有了这样一个单线程信息，也可以把这份数据集用于设计并测试一个优先邮件排序系统。我们也只关心这份数据集中的正常邮件，因此，所要检查的邮件都是用户希望出现在其收件箱中的。

然而，在处理之前，必须思考我们的数据集相比一份详尽的用户日志（比如Google所拥有的）有何不同，以及这如何影响了算法中能够使用的特征。首先来回顾一下Google提出的四类特征，并考虑其如何应用于我们的数据集。

警告： 一份详尽的邮件日志和我们要处理的数据之间最大的不同就是：我们只能看到接收邮件的内容本身。这意味着我们实际上是“摸黑赶路”，因为无法得知用户何时对邮件做了何种响应，也不知道一个线程是否由该用户发起。这是一个显著的局限性（significant limitation），因此在本章所使用的算法和策略应该仅仅视为练习，而不是在企业级智能收件箱系统实现的例子。我们想要达到的目的是给大家展示其原理，甚至在这样的局限性下，也能用这份数据来创建邮件重要性表征量并设计一个相当好的排序系统。

既然邮件是基于事务的媒介，那么其社交特征在邮件重要性评估方面举足轻重。在我们

的案例中，仅能得到一半的事务记录[译注3]，在特征详尽的案例中，会测量用户和不同发件人之间的交互量来确定哪一个发件人得到了该用户更多的立即响应。然而，在我们的数据中，仅仅能测量接收量。因此，可以假设这种单向度量能够较好地代表我们试图从数据中抽取出的社交特征类型。

很明显这并不理想。但是别忘了，在这个案例中只使用了SpamAssassin公开语料库中的正常邮件。如果一个人从一个固定地址收到了大量的正常邮件，那么可能的原因是发件人和该用户之间建立了较强的社交联系。也有可能是因为该用户加入了一个邮件量巨大的邮件列表中，那么他可能希望这样的邮件优先级不要太高。基于此，我们在开发排序系统时必须考虑结合其他特征来平衡这种类型的信息。

若只关注来自某一固定地址的邮件量，带来的一个问题就是延长了时间属性。跟一份详尽的日志相比，我们的数据集是静态的，因此必须把数据拆分成若干时间片段，然后测量每一个片段内的邮件量，从而能对时间动态性有更深入的理解。

在这个案例中，我们要对所有邮件信息简单地按时间排序，然后将数据集一分为二（这一点在稍后会详细讨论）。第一部分数据用于训练排序系统，另一部分数据用于测试排序系统。因此，在训练数据所覆盖的整个时间周期内，来自每一个地址的邮件量将用于训练排序系统的社交特征。

鉴于数据的先天不足，这样的处理可能开了个好头，但是如果希望排序更准确，就需要在理解上更深入些。为了对动态特征有更细粒度的认识，这就需要拆分数据，其方法之一就是，识别出会话线程，然后度量线程内的活动。（为了识别线程，可以从其他邮件客户端借鉴技术，匹配主题中的线程关键词项，比如“RE：”）尽管不知道用户对一个线程采取了哪些行为，但是我们假设：如果线程很活跃，那么它就比不活跃的线程更重要。通过把时间像这样切分成片段，对要用于邮件优先级建模的线程特征来说，可以获得更准确的表征量。

接下来，可以从邮件中抽取许多内容特征添加到特征集里。在这个案例中，像以前一样将问题简化，把第3章的文本挖掘技术沿用到这里。具体而言就是，如果有些词项同时出现在邮件主题和正文中，用户将来收到新邮件，当主题和正文中都包含这些词项时，它就比其他邮件要重要些。这是个常见的方法，在Google智能收件箱论文中也有所涉及。基于主题和正文包含的词项来引入内容特征，我们遇到了一个有趣的问题：权重（weighting）。在典型的邮件中，主题包含的词比正文要少得多，因此，同一个词项出现在主题时和出现在正文时，应该被赋予不同的权重值。

译注3：有收有发才是完整的事务记录，此处指“有收无发”的事务记录。

最后，在很多已发布的企业级智能收件箱的实现中还采用了许多其他特征（比如说Gmail），但在这个案例中根本得不到这些特征。前文提到过，在社交特征集方面，我们简直是两眼一抹黑，因此必须使用表征量来测量这些交互行为。此外，还有很多用户行为甚至都无法估计。比如说，用户进行标记和移动邮件的行为，我们完全不得而知。在Google的智能收件箱实现中，这些行为构成了其行为集合的重要部分，但是在本案例中完全缺失了。再强调一遍，虽然与使用详尽日志的方法相比，本案例中的方法因为没有详尽的日志记录，的确存在一个很大的缺陷，但是这一缺失并不影响我们的结果。

现在，我们对用于创建邮件排序系统的特征集有了一个基本蓝图。首先，对邮件按时间排序，因为在本案例中，我们感兴趣的大部分预测都包含在时间维度中。第一部分邮件用于训练排序算法。接下来，四种特征将用于训练过程。首先是替代社交特征的表征量，即衡量训练数据中来自某个用户的邮件量。接下来，通过发现不同线程来压缩时间测量值，并按照活跃度给线程排序。最后，基于邮件主题和正文中的高频词项增加两个内容特征。图4-1示意了从邮件中抽取这些特征的方法。

在下一节中，我们将实现这里所描述的智能收件箱。通过使用这里列出的特征，我们将制定一个权重计算方法，从而快速地把更重要的邮件放到栈顶。和之前一样，首先要读取原始邮件数据，根据特征集抽取相应片段。

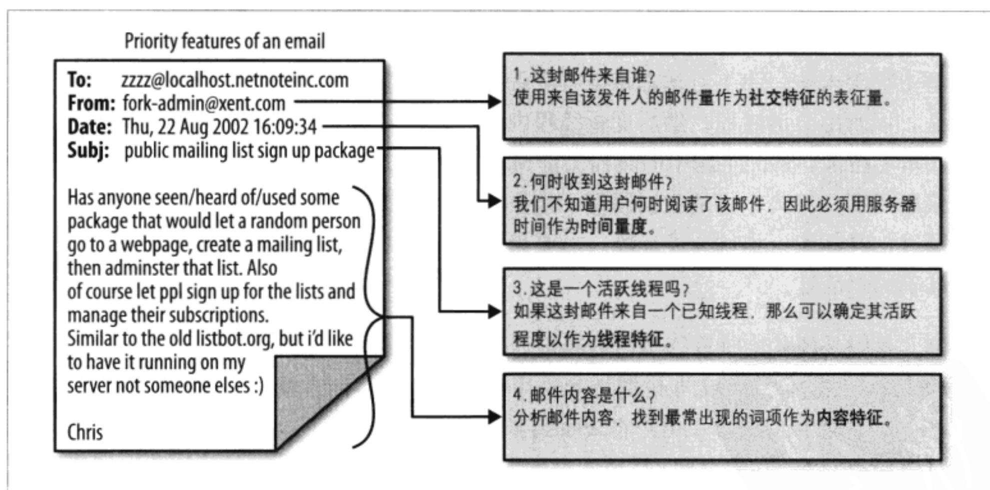


图4-1：从邮件数据中抽取优先级特征的策略

实现一个智能收件箱

到目前为止，你可能已经注意到一个趋势：在进入机器学习最迷人的地带之前，我们都

要在数据泥潭中打滚，忙着对数据进行拆分、抽取、解析直到其形式符合分析标准为止。到目前为止，所经历的过程都还算轻松。构建垃圾分类器，只需简单地抽取邮件正文，然后把所有重活都交给tm程序包去做。然而在本案例中，增加了若干其他特征到数据集中，并且引入了时间维度，使这个过程更复杂。因此，处理数据的工作量显著增加。但是，我们是黑客，因此充分和数据打成一团正合我意。

在这个案例中，只关注SpamAssassin公开语料库的正常邮件。和垃圾分类案例不同之处在于这个案例中我们不在意邮件的类型，而是重点研究如何对每封邮件按照优先级进行排序。因此，要用到数据量最大的易识别的正常邮件数据集，也不再担心会带入其他类型的邮件。因为完全可以假设用户在确定哪一封邮件优先级更高时不会费心区别邮件的类型[注1]，所以完全没有必要把类型信息引入到排序系统中来。相反，我们希望可以从一个用户的邮件中学习到尽可能多关于特征集的知识，这也是使用易识别为正常的邮件数据集的原因所在。

```
library(tm)
library(ggplot2)

data.path <- "../../03-Classification/code/data/"
easyham.path <- paste(data.path, "easy_ham/", sep="")
```

和第3章类似，本章要使用的第一个程序包就是tm，用于抽取主题和正文共同出现的词项；另一个就是ggplot2，用于将结果可视化。而且，因为SpamAssassin公开语料库是一份相当大的文本数据集，所以我们不会将其复制到本章的目录*data/*下，而是把相对路径设置为第3章的对应文件。

接下来，我们要实现一系列的函数，然后利用这些函数共同分析每一封邮件，得到如图4-1所示的特征集合。从这张图中可以看出，需要从每一封邮件中抽取四个元素：发件人的地址、接收日期、主题、邮件正文。

用于抽取特征集合的函数

大家应该还记得，第2章曾提出数据方块的概念。因此，在这个案例中，构建训练数据就是一个“方块化”的过程。我们需要把邮件数据塑造成为一个可用的特征集合。从邮件里抽取的特征组成了训练数据方块的每一列，而每一行则是来自每一封邮件的特征唯一值。用这种方式把数据概念化非常有用，因为我们需要将邮件中半结构化的文本数据转换为高度结构化的训练数据集，以便用于后续邮件排序。

```
parse.email <- function(path) {
```

注1：　简单地说，这个假设就是：对于更验以识别为正常的邮件，用户不会对其采取什么行为。

```
    full.msg <- msg.full(path)
    date <- get.date(full.msg)
    from <- get.from(full.msg)
    subj <- get.subject(full.msg)
    msg <- get.msg(full.msg)
    return(c(date, from, subj, msg, path))
}
```

为了对这个过程进行分析，我们退一步，先来了解一下parse.email函数。这个函数会调用一系列的辅助函数，从每一封邮件中抽取相应数据，然后依次放入一个向量中。命令c(date, from, subj, msg, path)创建的向量构成了数据中的一行，这个数据最终会构成训练数据。但是，把邮件转换为向量的过程需要一些经典的文本处理方法。

注意： 我们把路径作为数据的最后一列进行保存，这使测试阶段的排序更容易一些。

```
msg.full <- function(path) {
    con <- file(path, open="rt", encoding="latin1")
    msg <- readLines(con)
    close(con)
    return(msg)
}
```

如果你已经完成了第3章的案例，那么应该非常熟悉msg.full这个函数。在这个函数里，我们只是简单地打开一个文件的路径，并且读取文件内容到一个字符串向量中。readLines函数会生成一个向量，向量中的元素就是文件的每一行。与第3章不同的是，这里我们不再预处理数据，因为需要从邮件中抽取不同的元素。相反，我们会把整个邮件作为一个字符串向量返回，再另外写一些函数来处理这个向量，以抽取必需的数据。

有这个邮件向量在手，我们要在数据泥潭中奋力开辟一条道路，以便能够从邮件数据中抽取出尽量多的有用信息——并且用统一的方式将其组织好——从而构建出训练数据。首先进行相对简单的工作，抽取发件人地址。要完成这一任务——以及本节所有其他数据的抽取——我们需要找到邮件中的文本模式，因为利用文本模式可以识别出我们所要寻找的数据。既然如此，那先来看几封邮件，如例4-1所示。

例4-1：电子邮件的“来自”文本模式差异示例

```
Email #1
........................................................
X-Sender: fortean3@pop3.easynet.co.uk (Unverified)
Message-Id: <p05100300ba138e802c7d@[194.154.104.171]>
To: Yahoogroups Forteana <zzzzteana@yahoogroups.com>
From: Joe McNally< joe@flaneur.org.uk>
X-Yahoo-Profile: wolf_solent23
MIME-Version: 1.0
Mailing-List: list zzzzteana@yahoogroups.com; contact
  forteana-owner@yahoogroups.com
........................................................
```

```
Email #2
..................................................
Return-Path: paul-bayes@svensson.org
Delivery-Date: Fri Sep 6 17:27:57 2002
From: paul-bayes@svensson.org (Paul Svensson)
Date: Fri, 6 Sep 2002 12:27:57 -0400 (EDT)
Subject: [Spambayes] Corpus Collection (Was: Re: Deployment)
In-Reply-To: <200209061431.g86EVM114413@pcp02138704pcs.reston01.va.comcast.net>
Message-ID: <Pine.LNX.4.44.0209061150430.6840-100000@familjen.svensson.org>
..................................................
```

分析完几封邮件之后，我们可以观察到文本中发件人邮件地址的关键模式。例4-1展示的两段邮件摘录突出显示了这些模式。首先，我们需要识别每一封邮件中包含邮件地址的行。从上述示例中可以看出，这样的行总是包含“From:”这个词项，这在第3章所提到的邮件协议中已经指明。因此，我们可以利用这个信息来搜索由每封邮件得到的字符串向量，从而识别出正确的元素。但是，从例4-1中可以看出，不同的邮件地址写法有所不同。这一行总是包含发件人的名字和发件人的邮件地址，但是有时候把地址放在尖括号中（邮件Email #1），有时候没有放在尖括号中（邮件Email #2）。为此，我们要写一个get.from函数，使用正则表达式来抽取这个特征数据。

```
get.from <- function(msg.vec) {
    from <- msg.vec[grepl("From: ", msg.vec)]
    from <- strsplit(from, '[":<> ]')[[1]]
    from <- from[which(from !="" & from !=" ")]
    return(from[grepl("@", from)][1])
}
```

我们已经见识过，R有许多强大的函数是通过正则表达式来实现的。grepl函数的功能与标准的grep函数一样，用于匹配正则表达式模式串，只不过字母“l”代表“逻辑的”（logical）。因此，它不是返回向量的索引号，而是返回一个和msg.vec长度一样的向量，这个向量中的元素用于标识字符串向量每个元素是否匹配上模式串。在函数第一行后面，from变量是一个字符串向量，它只有一个元素：例4-1中突出显示的包含“From：”的那一行。

既然已经得到正确的行，就需要只抽取地址本身。为此，要用到strsplit函数，它根据给定的正则表达式模式把一个字符串拆分成一个列表。为了能正确地抽取出地址，需要考虑到例4-1中文本模式的差异性。于是，用方括号为模式创建一个字符集。这里用来拆分文本的字符包括：冒号、尖括号、空格。这种模式总是把地址拆分为列表的第一个元素，因此可以用[[1]]把地址从列表中抽取出来。然而，因为文本模式存在变化，所以会抽取到空元素。为了只返回邮件地址本身，我们会忽略这些空元素，然后查找包含“@”符号的元素并返回它们。目前我们已经分析得到了用于产生训练数据所需数据量的四分之一。

```
get.msg <- function(msg.vec) {
```

```
    msg <- msg.vec[seq(which(msg.vec == "")[1] + 1, length(msg.vec), 1)]
    return(paste(msg, collapse="\n"))
}
```

接下来抽取的两个特征：邮件主题和正文，操作相当简单。在第3章中，我们需要抽取正文来量化垃圾邮件和正常邮件中的词项。因此，那里的get.msg函数可以在这里的任务中简单复用。回想一下，邮件正文总是出现在邮件中第一个空行后面。因此，简单地在msg.vec中查找第一个空行即可，然后返回空行之后的所有元素。为了简化文本挖掘过程，用paste函数把这些向量转换成一个字符串元素的向量后返回。

```
get.subject <- function(msg.vec) {
    subj <- msg.vec[grepl("Subject: ", msg.vec)]
    if(length(subj) > 0) {
        return(strsplit(subj, "Subject: ")[[1]][2])
    }
    else {
        return("")
    }
}
```

抽取邮件主题和抽取发件人地址方法类似，实际上更简单些。在get.subject函数中，再一次通过grepl函数在每一封邮件中查找"Subject:"模式串，从而找出包含主题的行。但是，有这样一个问题：实际上数据集中并非每一封邮件都有主题。模式匹配在这样的极端情况下就会出现问题，为了防止出现这种问题，进行一个简单测试，看看grepl函数实际上是否返回了东西。这就需要检查subj的length（长度）是否大于0。如果大于0，把这一行按照模式拆分，并返回拆分得到的第二个元素；如果不大于0，则回一个空字符。在R中，当诸如grepl这样的匹配函数得不到匹配时，默认会返回一个指定的值，如integer(0)或者character(0)，这些值的长度为0。因此在一大堆杂乱的数据上运行函数时，进行这样的检查还是挺不错的。

警告： 在第3章的*code/data/hard_ham/*文件夹下，你看到的00175.*这个文件就是一封有问题的电子邮件。当试着给自己研究的问题引入数据时，像这样遇到异常数据的情况是再经常不过了。想要解决这一问题，需要反复尝试，就像我们在这里所做的一样。首要的问题是保持冷静，深挖数据以找到问题所在。在把数据解析成可使用形式的过程中，你若一帆风顺，那么很有可能你的做法不对。

到现在为止，我们已经得到了四分之三的特征了，不过，正是最后这个特征（邮件接收日期和时间）才最让我们痛不欲生。这个问题不好对付的原因有两个。第一个原因，处理日期是一个永远的难题，因为不同的编程语言常常在时间的设计理念上有些许不同，在这个案例中，R也不例外。最终我们想把日期字符串转换成POSIX日期对象，目的是可以按照日期对数据排序。但是要做到这一点，需要一个统一的日期表现形式，这也是

要提到的第二个原因：在SpamAssassin公开语料库中，邮件接收日期和时间表现形式存在很大的差异。例4 2展示了一些这方面的差异。

例4-2：邮件接收日期和时间表现形式差异

```
Email #1
........................................................
Date: Thu, 22 Aug 2002 18:26:25 +0700

  Date: Wed, 21 Aug 2002 10:54:46 -0500
  From: Chris Garrigues lt;cwg-dated-1030377287.06fa6d@DeepEddy.Comgt;
  Message-ID: lt;1029945287.4797.TMDA@deepeddy.vircio.comgt;
........................................................

Email #2
........................................................
List-Unsubscribe: lt;https://example.sourceforge.net/lists/listinfo/sitescooper-talkgt;,
  lt;mailto:sitescooper-talk-request@lists.sourceforge.net?subject=unsubscribegt;
List-Archive: lt;http://www.geocrawler.com/redir-sf.php3?list=sitescooper-talkgt;
X-Original-Date: 30 Aug 2002 08:50:38 -0500
Date: 30 Aug 2002 08:50:38 -0500
........................................................

Email #3
........................................................
Date: Wed, 04 Dec 2002 11:36:32 GMT
Subject: [zzzzteana] Re: Bomb Ikea
Reply-To: zzzzteana@yahoogroups.com
Content-Type: text/plain; charset=US-ASCII
........................................................

Email #4
........................................................
Path: not-for-mail
From: Michael Hudson lt;mwh@python.netgt;
Date: 04 Dec 2002 11:49:23 +0000
Message-Id: lt;2madyyyyqa0s.fsf@starship.python.netgt;
........................................................
```

你也看到了，当我们从每一封邮件中抽取日期和时间信息时，需要考虑多方面的因素。从例4-2中可以观察到第一个特点就是：我们所要抽取的数据总是包含“Date：”，但是，在使用这个模式时必须注意可能存在的问题。例4 2中的邮件Email #1表明有时候会有多行匹配上这个模式。同样邮件Email #2也体现了一个特征：有的行可能会部分匹配。这两种情况中的任何一种发生时，这些行的数据就会互相冲突。第二个特点是：在这四个邮件示例中，我们看到，日期和时间都不是用统一方式表示的。在所有邮件中，都有一个附加的格林尼治标准时间（GMT）偏移量，以及一些其他的标签信息。最后一点，Email #4中日期和时间的格式和前面两封邮件中的完全不同。

在把数据转换成统一的、可用形式的过程中，所有这些信息尤为关键。然而，现在我们要定义一个get.date函数，剔除附加的时间偏移信息，只抽取日期和时间信息。

一旦得到了所有日期/时间字符串之后，我们就要着手处理形式各异的日期/时间格式，把它们转换成统一的POSIX对象，但是这一步并不在get.date函数中完成。

```
get.date <- function(msg.vec) {
```

```
    date.grep <- grepl("^Date: ", msg.vec)
    date.grepl <- which(date.grep == TRUE)
    date <- msg.vec[date.grep[1]]
    date <- strsplit(date, "\\+|\\-|: ")[[1]][2]
    date <- gsub("^\\s+|\\s+$", "", date)
    return(strtrim(date, 25))
}
```

我们之前提到过，许多邮件会有多行匹配上或者部分匹配上模式“Date:”。然而，请注意在例4-2的邮件Email #1和Email #2中，只有一行所包含的“Date:”是在字符串首部。在邮件Email #1中，在模式串之前存在若干空字符，在邮件Email #2中，模式串是“X-Original-Date:”的一部分。可以要求正则表达式只匹配在字符串首部的“Date:”，这要使用脱字符，用“^Date:”来完成。现在grepl只有在向量元素的首部发现模式串时才返回TRUE。

接下来，我们想返回msg.vec中匹配上模式串的第一个元素。也许我们可以简单地返回msg.vec中匹配上grepl中的模式的元素，但是，如果一封邮件正文中有一行是以“Date:”开始的那怎么办？如果这种异常情况发生了，我们也知道，第一个匹配上的元素是在邮件的头部信息中，因为头部信息总是在邮件正文之前。为了防止这种情况出现，我们总是返回第一个匹配的元素。

现在需要处理这一行文本，目的是返回一个字符串，并可以最终转换为R中的POSIX对象。我们已经注意到日期和时间后存在一些多余的信息，并且它们的格式不统一。为了把日期和时间信息分离出来，需要用字符把字符串进行拆分，从而标记出多余信息。用于拆分的字符，可能是冒号、加号或者减号。在大多数情况下，它们会把日期和时间信息保留下来，还会保留一些其后的空白字符。接下来这一行中的gsub函数会把首部或者尾部的空白字符替换掉。最后，对于例4 2的Email #3中看到的那种多余数据，我们会简单地把25个字符之后的裁剪。一个标准的日期/时间字符串是25个字符长度，因此可知第25个字符之后任何的字符都是多余的。

```
easyham.docs <- dir(easyham.path)
    easyham.docs <- easyham.docs[which(easyham.docs != "cmds")]
    easyham.parse <- lapply(easyham.docs, function(p) parse.email(paste(easyham.path,
                                                                      p, sep="")))

ehparse.matrix <- do.call(rbind, easyham.parse)
allparse.df <- data.frame(ehparse.matrix, stringsAsFactors=FALSE)
names(allparse.df) <- c("Date", "From.EMail", "Subject", "Message", "Path")
```

恭喜！你成功地把这批杂乱的邮件数据集转换成了一个结构化的数据方块，可用于训练我们的排序算法了。现在我们还得实现一个启动开关。与在第3章中所做的类似，创建一个向量，包含所有“易识别的正常邮件”的文件路径，并把其中多余的“cmds”文件路径从向量中移除，然后使用lapply函数对每一个邮件文件应用parse.email函数。因为

我们要使用的是前一章的数据路径，所以也必须要在lapply函数内部用paste函数把文件的相对路径和easyham.path变量连接起来。

接下来，我们需要把lappy函数返回的向量列表转换成一个矩阵——也就是数据方块。和以前一样，我们会用到do.call函数，同时结合rbind，用于创建ehparse.matrix对象。然后把它转换成一个包含多个字符串向量的数据框，并把每一列的名称设置成c("Date", "From.EMail", "Subject", "Message","Path")。用命令head(allparse.df)显示数据框的前几条数据，检查一下结果，为了节省篇幅，这里就不再展示这个过程了，但是建议你亲自实践一下。

在为这份数据指定权重计算策略之前，仍有一些琐碎的工作需要做。

```
date.converter <- function(dates, pattern1, pattern2) {
    pattern1.convert <- strptime(dates, pattern1)
    pattern2.convert <- strptime(dates, pattern2)
    pattern1.convert[is.na(pattern1.convert)] <-
    pattern2.convert[is.na(pattern1.convert)]
    return(pattern1.convert)
}

pattern1 <- "%a, %d %b %Y %H:%M:%S"
pattern2 <- "%d %b %Y %H:%M:%S"

allparse.df$Date <- date.converter(allparse.df$Date, pattern1, pattern2)
```

前面已经说过，我们抽取日期所做的第一步只是简单地分离出文本。现在要把这些文本转换成POSIX对象，以便可以进行逻辑比较。这一步是必须的，因为我们需要按照时间对邮件进行排序。还记得吗？我们说过完成这个案例的关键就是时间维度，以及如何将观测特征之间的时间差异用于衡量邮件重要性。因此，字符串形式的日期和时间是无法满足这些要求的。

在例4-2中可以看到，日期格式有两种。在这些例子中，Email #3的日期/时间格式是“Wed, 04 Dec 2002 11:36:32”，而Email #4的格式是“04 Dec 2002 11:49:23”。为了把这两种字符串转换成POSIX格式，要用到strptime函数，但是需要传入两种不同的日期/时间格式来完成转换。每一个日期/时间字符串都匹配一个具体的POSIX格式，因此，要指明转换字符串符合哪一种格式。

注意： R用标准POSIX日期/时间格式串来完成这些转换。这些格式字符串有许多选项，推荐通读strptime函数的整个文档，用命令?strptime即可看到所有选项。我们这里仅仅用到了很少一部分，但是更深入地理解它们非常有助于今后用R处理日期和时间。

我们需要用两种不同的POSIX格式分别转换allparse.df中Date这列的字符串，然后将

这两种格式组合加入数据框中完成转换。为实现这一步，我们定义了date.converter函数，用于输入两个不同的POSIX模式串以及一个日期字符串向量。当传给strptime的模式串并不匹配传给它的日期字符串时，默认返回NA值。我们可以利用这个值，把第一次转换得到NA值的元素用第二次转换的结果替代，从而将两个转换结果结合起来。因为我们知道数据中只存在两种日期模式，所以得到的结果就只是一个向量，其中包含所有转换为POSIX对象的日期字符串。

```
allparse.df$Subject <- tolower(allparse.df$Subject)
allparse.df$From.EMail <- tolower(allparse.df$From.EMail)

priority.df <- allparse.df[with(allparse.df, order(Date)),]

priority.train <- priority.df[1:(round(nrow(priority.df) / 2)),]
```

清洗工作的最后一小步是把主题（Subject字段）和发件人（From字段）这两列的字符串向量全部转换为小写。再强调一次，这样做的目的是保证在训练阶段所有数据条目的格式尽量统一。接下来，我们结合使用with和order命令按照时间对数据进行排序。（R的排序方法不是很直观，但是你会发现将经常使用这种简便写法，因此最好熟悉它。）结合使用两个命令后会返回一个向量，向量元素为按照时间升序排列的数据索引号。然后，用这些索引号对数据框进行排序，我们需要按照这个顺序引用allparse.df的元素，并且在收尾的方括号前加上一个逗号，这表示所有列都按照这个方式进行排序。

最后，我们把按照时间排序的数据框前半部分存储到priority.train中。这部分数据用于训练排序算法。稍后我们要用priority.df的第二部分数据来测试排序算法。鉴于数据已经结构化完毕，我们准备开始设计排序策略。既然已有了特征集，处理方式之一就是对训练数据的每一个观测特征赋予一定的权重。

设计用于排序的权重计算策略

在实现权重计算策略之前，请容我们简短地闲话几句关于尺度的问题。稍微想一想你自己的邮件行为。你是否经常和基本固定的一些人互动？你清楚自己一天要收多少封邮件吗？你一周会收到多少封陌生人的邮件？如果你和我们没什么不同，那么我们猜想你的邮件行为和其他大多数人一样，大致符合2/8准则。也就是说，你80%的邮件行为是和通讯录中20%的人之间发生的。那么，为什么这很重要？

我们需要设计一个策略来对数据中特定的观测特征加权，但是因为这些特征的观测数据在尺度上可能存在不同，所以不能直接进行比较。更准确地说，我们不能直接比较它们的绝对值。就拿已经解析好的训练数据来说，我们知道排序算法的特征之一就是社交行为，可以用训练数据中来源于每个地址的邮件数量来近似表示这个特征。

```
from.weight <- ddply(priority.train, .(From.EMail), summarise, Freq=length(Subject))
```

要分析尺度问题如何影响第一个特征，需要统计每个邮件地址出现在数据中的频数。要用plyr程序包来完成这一步，这个包已经在加载ggplot2时作为依赖包加载了。如果你已经完成了第1章的案例，那么你已经见识过plyr的功能了。简单地说，plyr中的函数家族用于把数据分解成一些小片段，从而可以在这些片段上一次性完成操作。（这非常类似于流行的Map-Reduce模式，该模式适用于很多大规模数据分析环境。）这里我们完成的任务很简单：在From.Email列中找出所有包含地址的行，并统计其数量。

这里用到了ddply函数，它在包含训练数据的数据框上执行操作。语法要求我们先定义想要操作的数据组——这里就只是From.Email这个维度——然后定义在这个组内执行的操作过程。这里我们还用到了summarise选项来创建名为Freq的新列，用于保存频次信息。你可以用命令head(from.weight)查看结果。

注意： 在这个案例中，操作会询问分解后的数据框中列向量Subject的长度，但是实际上可以使用训练数据中任何列的名称来得到同样的结果，因为所有符合标准的数据都具有相同的长度。越熟悉如何用plyr来操作数据，就越有助于你进阶深入，同时强烈推荐阅读程序包作者的论文[HW11]。

为了更好地理解这份数据的尺度，我们把结果绘制出来了。图4-2所示的是发送邮件数量在6封以上的发件人邮件量条形图。为了方便看图，我们对数据进行了截断，即便如此，我们仍然可以看到数据尺度增长之迅速。发送量排在第一位的*tim.one@com-cast.ent*在我们的数据中共发送了45封，这是训练数据中一般人发送量的15倍！但是*tim.one@com-cast.ent*相当独特，你可以从图4-2中看出，仅有几个发件人的数量能望其项背，其他人的邮件量急剧下降。我们怎么才能计算一般人的观测数据权重值，而不会因为个别特殊情况，比如此处发件最多的人，而导致这些权重值偏移？

Log加权策略

解决的方法就是尺度变换。我们需要让特征的数值关系不那么极端，如果只比较频次的绝对值，那么一封来自*tim.one@com-cast.ent*的邮件就比一般人发送的重要15倍。这个就有问题了，因为我们想基于排序算法在学习阶段产生的权重值范围来确定一个阈值，用于判断一封邮件是否排序优先。若是偏斜太过极端，阈值就会要么太低要么太高，考虑到此处数据的本质属性，我们需要重新设定各统计值的尺度。

于是引入了对数（logarithms）和对数变换（log-transformations）。你可能在初等数学中就已熟悉对数，如果不熟悉也无妨，这个概念也很简单：给定一个数，用对数函数可以得到一个指数，这个指数满足以下等式：某个底数可以由这个指数提升到给定的那个数。

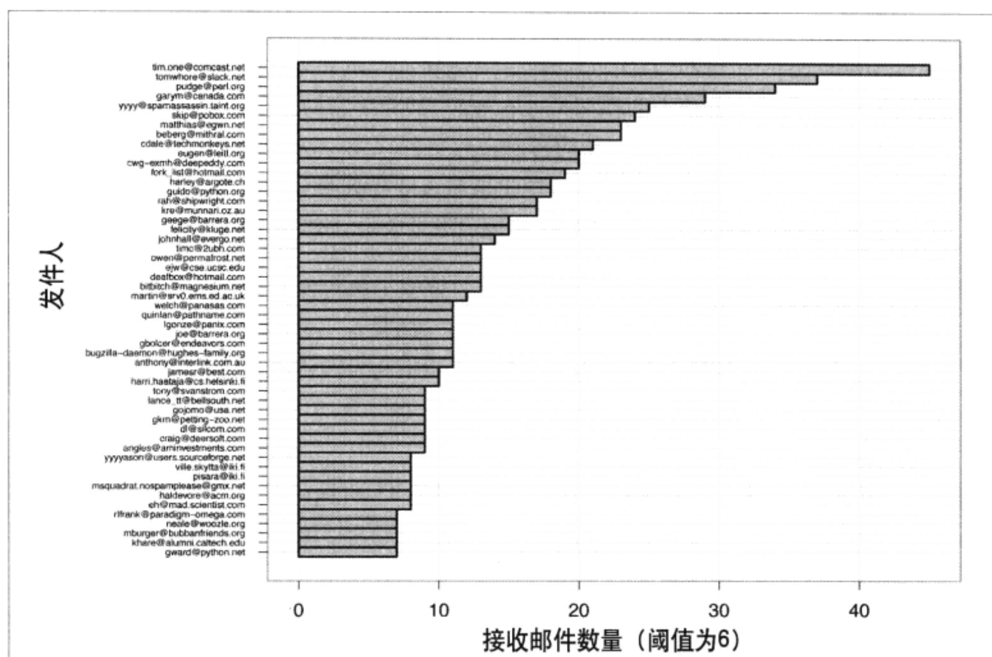


图4-2：训练数据中从不同发件人接收到的邮件数量

这个底数在对数变换中很关键。作为一名黑客，我们的思维模式更熟悉二进制的事物，或者二元（binary）的事物。我们可以毫不费力地构建一个底数为2的对数变换。这样就需要解一个关于这个指数值的方程：底数2的该指数次幂等于要变换的输入值。举个例子，如果以底数为2进行16的对数变换，那就得到4，因为2的4次幂等于16。从本质上说，我们在进行指数函数的逆运算，因此这种变换在数据符合指数函数时效果最好。

两个最常见的对数变换就是所谓的自然对数（natural log）变换和常用对数（log base-10）变换。前者底数是一个特殊的数e，一个值约为2.718的无理数（像π一样）。自然对数常表示为ln。用这个常数变换的尺度在自然界中随处可见，事实上，我们可以通过几何方式生成这条曲线，就是一个在圆内关于角度的函数[译注4]。你可能很熟悉一些符合自然对数的形状或者关系，尽管也许你并没有以这种角度思考过这些现象。图4-3展示了一个自然对数螺旋，这个螺旋可见于许多自然界现象中，比如说鹦鹉螺的内部结构、飓风（或者龙卷风）的螺旋状。甚至银河系中星际颗粒的散布也遵循自然对数螺旋。另外，许多专业相机设置的孔径也是按照自然对数变化的。

译注4：这条曲线称为对数螺旋曲线，又称为等触螺线，在极坐标系（r, θ）中，其解析式为 $r=ae^{b\theta}$。

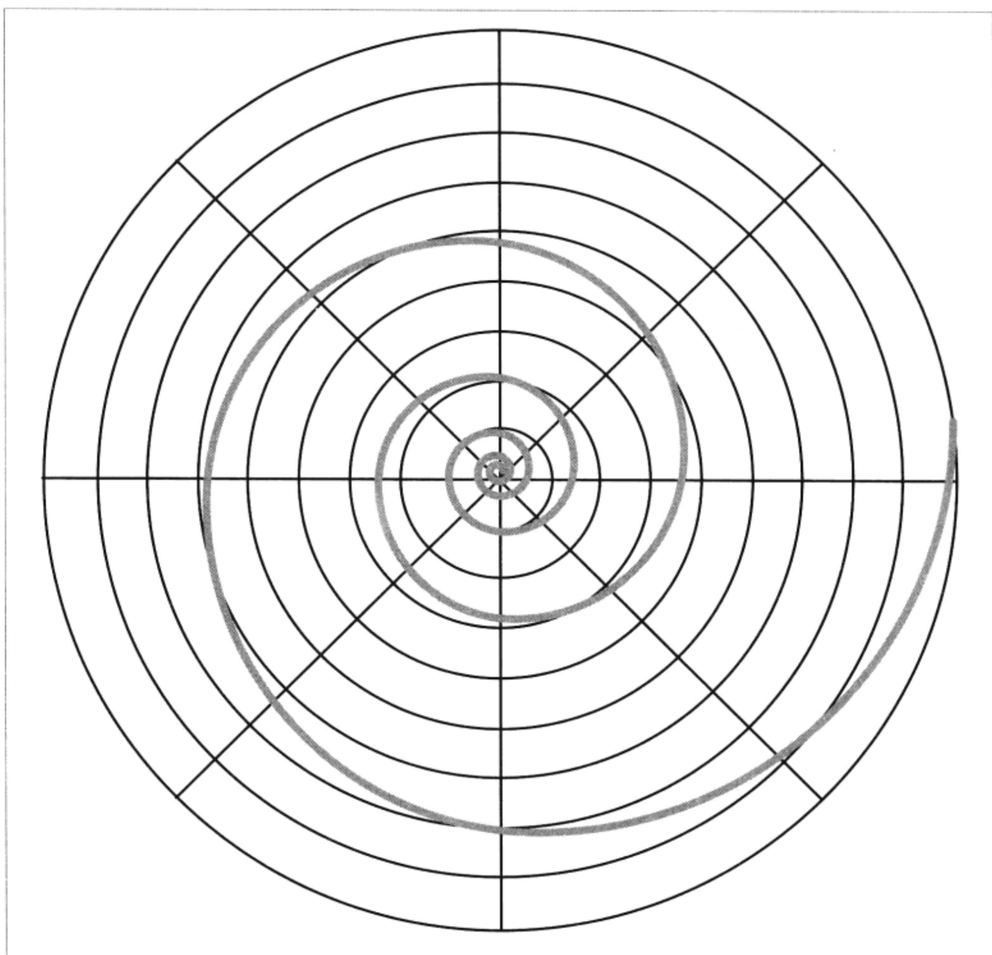


图4-3：在自然界随处可见的自然对数螺旋

既然自然对数和这么多自然现象都相近，那么能把社交数据（诸如邮件行为）重新缩放的指数函数还真是了不起。另外，常用对数变换也常常表示为log10，就是把自然对数中的e换为10即可。认识了对数变换的原理之后，我们知道和自然对数相比，常用对数会把值变换得更小。举个例子，1000进行常用对数变换是3，因为10的3次幂是1000，而自然对数变换大约是6.9。因此在数据尺度的指数形状非常陡时使用常用对数变换更合理。

邮件量数据的这两种转换方式在图4-4中均有展示。在图中，我们看到训练数据中用户发送邮件量的指数形状非常陡峭，通过分别采用自然对数和常用对数变换后，曲线明显平缓了。我们已经知道，常用对数变换的程度更大，而自然对数变换则仍然保留了一些差

异性，可以帮助我们从这份训练数据中得到有意义的权重值。因此，我们用自然对数来定义邮件量这个特征的权重。

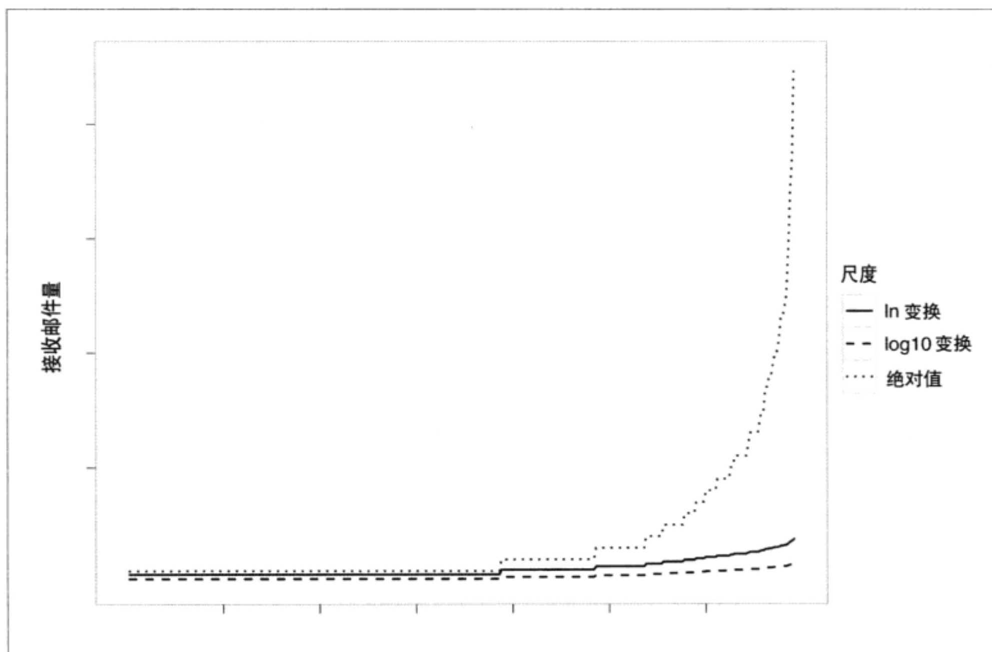


图4-4：接收邮件量分别为绝对值、ln变换值以及log10变换值的区别

注意： 需要强调的一点是，就像我们这里所做的，以及在第2章详细阐释过的一样，在处理任何机器学习问题时，对数据进行可视化分析绝对是一个好办法。我们想知道特征集合的各个观测值之间的关系如何，以便能做出最好的预测。通常达到这一目的最好的办法便是对数据可视化。

```
from.weight <- transform(from.weight, Weight=log(Freq + 1))
```

最后，回想一下，在初等数学中学过的指数规则告诉我们，任何值的0次幂都是1。在使用对数变换计算权重时记住这一点非常重要，因为观测值为1时，就会变换成0。在计算权重时这会出现问题，因为用0乘以其他特征权重值时会得到0。为了避免出现这种情况，我们在对数变换前总是对任何观测值都加1。

注意： 实际上R中的函数`log1p`就是计算`log(1+p)`的，但是本着学习就要知其所以然的目的，我们还是“手工”加1。

进行这样的尺度缩放，并不会影响结果，而且还会保证所有权重大于0。在这里，我们使用了log函数的默认底数，即自然对数。

警告： 出于我们的目的，绝不允许特征集合中出现值为0的观测记录，因为这是在统计数量。如果某些特征不存在观测记录，那么它也不会出现在我们的训练集中。但是，在某些情况下，这也不对，还是可能会在数据中出现值为0的观测记录。0的对数是无定义的，而如果非要用R去计算的话，会返回一个特殊的值-Inf（负无穷）。如果数据中出现-Inf值常常会破坏整个结果。

邮件线程活跃度的权重计算

从邮件中抽取的第二个特征就是线程行为。之前已经解释过，我们给这些用户构建排序算法，却没法知道用户是否回复过邮件。不过，可以按照线程对邮件进行分组，然后衡量每个线程开始之后的活跃程度。再强调一次，构建这个特征集的假设还是时间很重要，于是，在短时间内有更多邮件发送的线程就更活跃，也因此更重要。

在我们这份数据中，邮件并没有明确的线程ID，但是在训练数据中识别线程的一种逻辑方式是查找具有共同主题的邮件。例如，如果发现一个主题以“re:”开始，那么我们就知道这是某个线程的一部分。当发现像这样的邮件时，可以再找一下在这个线程里面的其他邮件，并测量其活跃度。

```
find.threads <- function(email.df) {
    response.threads <- strsplit(email.df$Subject, "re: ")
    is.thread <- sapply(response.threads, function(subj) ifelse(subj[1] == "", TRUE,
                        FALSE))
    threads <- response.threads[is.thread]
    senders <- email.df$From.EMail[is.thread]
    threads <- sapply(threads, function(t) paste(t[2:length(t)], collapse="re: "))
    return(cbind(senders,threads))
}

threads.matrix<-find.threads(priority.train)
```

这正是函数`find.threads`要对训练数据所做的处理。如果把训练数据中的主题都用“re:”进行拆分，那么可以通过寻找拆分后第一个元素是空字符的字符串向量来找到各个线程。一旦知道了训练数据中哪些是线程的一部分，就可以从这些线程中抽取出主题和发件人。这样，结果矩阵中就包含训练数据中的所有发件人和初始线程的主题。

```
email.thread <- function(threads.matrix) {
    senders <- threads.matrix[, 1]
    senders.freq <- table(senders)
    senders.matrix <- cbind(names(senders.freq), senders.freq, log(senders.freq + 1))
    senders.df <- data.frame(senders.matrix, stringsAsFactors=FALSE)
    row.names(senders.df) <- 1:nrow(senders.df)
```

```
    names(senders.df) <- c("From.EMail", "Freq", "Weight")
    senders.df$Freq <- as.numeric(senders.df$Freq)
    senders.df$Weight <- as.numeric(senders.df$Weight)
    return(senders.df)
}

senders.df <- email.thread(threads.matrix)
```

接下来，根据线程中最活跃的发件人来赋予权重，该权重会增加在整个训练数据集上得到的基于邮件数量的权重，不过现在只关注出现在threads.matrix中的发件人。email.thread函数的输入是threads.matrix，然后生成第二个基于数量的权重。这和前一节所做的事情很类似，不同的是这里要用到table函数统计发件人在线程中出现的频率。这一做法仅仅是为了展示R中完成同样计算的不同方法，这里是在矩阵上完成，而不是用plyr在数据框上完成。这个函数的大部分工作都是在对数据框senders.df做些琐碎处理，不过，请注意这里再一次用到了自然对数来赋予权重。

作为线程特征最后一个组成部分的权重，我们会基于已知的活跃线程来构建。我们已经识别出训练数据中所有的线程，并且基于线程中的词项计算出了一个权重。现在我们想利用这个知识，给已知的活跃线程追加一个权重。这里存在一个假设，如果这个线程已知，那么我们认为用户会觉得这些更活跃线程更重要一些。

```
get.threads <- function(threads.matrix, email.df) {
    threads <- unique(threads.matrix[, 2])
    thread.counts <- lapply(threads, function(t) thread.counts(t, email.df))
    thread.matrix <- do.call(rbind, thread.counts)
    return(cbind(threads, thread.matrix))
}

thread.counts <- function(thread, email.df) {
    thread.times <- email.df$Date[which(email.df$Subject == thread
    | email.df$Subject == paste("re:", thread))]
    freq <- length(thread.times)
    min.time <- min(thread.times)
    max.time <- max(thread.times)
    time.span <- as.numeric(difftime(max.time, min.time, units="secs"))
    if(freq < 2) {
        return(c(NA,NA,NA))
    }
    else {
        trans.weight <- freq / time.span
        log.trans.weight <- 10 + log(trans.weight, base=10)
        return(c(freq,time.span, log.trans.weight))
    }
}

thread.weights <- get.threads(threads.matrix, priority.train)
thread.weights <- data.frame(thread.weights, stringsAsFactors=FALSE)
names(thread.weights) <- c("Thread","Freq","Response","Weight")
thread.weights$Freq <- as.numeric(thread.weights$Freq)
```

```
thread.weights$Response <- as.numeric(thread.weights$Response)
thread.weights$Weight <- as.numeric(thread.weights$Weight)
thread.weights <- subset(thread.weights, is.na(thread.weights$Freq) == FALSE)
```

我们要使用刚刚创建的threads.matrix在训练数据中查找每一个线程中出现的所有邮件。先创建get.threads函数，其参数是threads.matrix和训练数据。我们用unique命令创建了一个包含数据中所有线程主题的向量。现在，要利用这个信息并衡量每一个线程的活跃度。

这个任务由thread.counts函数完成。利用线程主题和训练数据作为参数，收集thread.times向量中线程所匹配的所有邮件日期和时间戳。可以计算出在训练数据中这个线程接收了多少封邮件，方法就是计算thread.times的长度。

最后，计算活跃度，需要知道这个线程在训练数据中存在了多长时间。虽然不那么明显，但是数据两端都存在一定的截断。也就是说，某个线程可能在开始收集训练数据之前或者在数据收集完成之后还收到了邮件。对此我们无能为力，因此就给每个线程设置了一个最近时间和最远时间，以此来计算线程的时间跨度。函数difftime会用指定单位计算两个POSIX对象的相隔时间。在我们的案例中，要用的是最小时间单位“秒”。

由于存在这个时间截断，可能会在一个线程中只有一封邮件的记录。这种情况可能是在训练数据刚开始收集时线程正好结束，也可能是数据收集结束之际线程才开始。在基于这个时间段上的线程活跃度计算权重之前，我们必须把这些只有一封邮件的线程标记出来。thread.counts函数末尾的if语句就是在做这件事，如果当前线程只有一封邮件它就返回一个全是NA元素的向量。稍后，我们会利用这个向量来把这些只有一封邮件的线程从活跃度权重数据中剔除。

最后一步就是为所有我们能测量的邮件[译注5]赋予权重。首先，计算线程在单位时间内邮件的到达率（每秒收到邮件数的均值），所以，如果一个线程内每秒发一封邮件，那么这个结果就是1。当然，实际中这个值要比1小得多：每个线程收到邮件的平均数大约是4.5，平均间隔时间是31 000秒（8.5小时）。在这个尺度下，大多数到达率都是很小的小数。还是老办法，用log转换这些值，但是因为要转换的是小数，所以会得到负值。我们不能在权重计算中引入负的权重，因此不得不进行一项附加变换，正式的叫法是仿射变换（affine transformation）。

仿射变换就是在空间里对点进行简单的线性移动。想象一下，在一张坐标纸上画一个正方形，如果你想移动正方形到纸上另一个位置，可以定义一个函数，把正方形上的所有点沿着同一个方向移动，这就是仿射变换。要让log.trans.weight得到正权重，我们简单地给所有转换值加10。这会保证所有的值都是一个合适的正值。

译注5：此处指频次大于1的线程邮件。

一旦用get.threads和thread.counts生成了权重数据，和从前一样，给thread.weights数据框做一些琐碎的处理，目的是与其他权重数据框中的名称保持一致。在最后一步里，我们用subset函数移除了所有指向只有一封邮件的线程记录（即被时间截断的线程）的行。我们现在用head(thread.weights)检查一下结果。

```
head(thread.weights)
                                      Thread   Freq  Response    Weight
1   please help a newbie compile mplayer :-)      4     42309  5.975627
2                prob. w/ install/uninstall      4     23745  6.226488
3                      http://apt.nixia.no/     10    265303  5.576258
4         problems with 'apt-get -f install'      3     55960  5.729244
5                  problems with apt update      2      6347  6.498461
6 about apt, kernel updates and dist-upgrade      5    240238  5.318328
```

前面两行就是线程活跃度权重计算策略的良好范例。这两个线程中的邮件数量都是4，然而线程prob. w/ install/uninstall在这份数据中的持续时间差不多是另一个的一半。根据我们的假设，这个线程的邮件更加重要一些，因此具有更高的权重。在这个案例中，我们赋予这个线程中邮件的权重是线程please help a newbie compile mplayer :-)中邮件的1.04倍。这在你看来可能合理，也可能不合理，这也是把设计和应用策略通用化过程中存在的问题。在这个案例中，我们的用户可能不会用这种方式来量化事物，而其他人可能会，但是因为我们想寻求一个通用解决方案，所以必须接受假设的结果。

```
term.counts <- function(term.vec, control) {
    vec.corpus <- Corpus(VectorSource(term.vec))
    vec.tdm <- TermDocumentMatrix(vec.corpus, control=control)
    return(rowSums(as.matrix(vec.tdm)))
}

thread.terms <- term.counts(thread.weights$Thread,
    control=list(stopwords=stopwords()))
thread.terms <- names(thread.terms)

term.weights <- sapply(thread.terms,
    function(t) mean(thread.weights$Weight[grepl(t, thread.
    weights$Thread,fixed=TRUE)]))
term.weights<-data.frame(list(Term=names(term.weights), Weight=term.weights),
    stringsAsFactors=FALSE, row.names=1:length(term.weights))
```

最后一份通过线程产生的权重数据就是这些线程中高频词项的权重。和第3章所做的事类似，我们创建一个通用函数term.counts，以一个词项向量和TermDocumentMatrix（TDM，词项文档矩阵）的选项列表为输入，产生词项的TDM，并且抽取所有线程中的词项频次。创建这份权重数据的假设是：出现在活跃线程邮件主题中的高频词比低频词和出现在不活跃线程中的词项更重要。我们在努力为排序算法加入尽可能多的信息，只为构建出较细粒度的邮件分层。为此，就不能只盯着早已活跃的线程，而是也想对那

些“貌似”先前活跃过的线程赋予一定权重，词项加权（term weighting）便是完成这件事的方式之一。

```
msg.terms <- term.counts(priority.train$Message,
    control=list(stopwords=stopwords(),
    removePunctuation=TRUE, removeNumbers=TRUE))
msg.weights <- data.frame(list(Term=names(msg.terms),
    Weight=log(msg.terms, base=10)), stringsAsFactors=FALSE,
    row.names=1:length(msg.terms))

msg.weights <- subset(msg.weights, Weight > 0)
```

最后一份权重数据的产生要基于训练数据的所有邮件中的词项频数。要进行的处理过程和我们在垃圾分类案例中所做的统计词频几乎相同，然而，这里要取词频的对数变换值作为词的权重值。因为有线程主题的词项频率权重，所以在数据框msg.weights中存在一个隐含的假设：和我们已读邮件相似的新邮件比那些完全陌生的邮件更重要。

现在，我们已有五项权重数据框，可以用于排序了！这些权重数据分别是：from.weight（社交特征）、senders.df（发件人在线程内的活跃度）、thread.weights（线程的活跃度）、term.weights（活跃线程的词项）以及msg.weights（所有邮件共有词项）。现在可以在训练数据上运行排序算法，找到一个可用于标记邮件是否重要的阈值。

训练和测试排序算法

为了给训练数据中每一封邮件产生一个优先等级，必须将前几节生成的所有权重相乘。这意味着对训练数据中每一封邮件都要解析邮件，抽取特征，然后用特征去匹配对应的权重数据框并查找其权重值。用这些权重值的乘积为每一封邮件产生单个且独一无二的排序值。下面的rank.message函数功能就是输入一封邮件的路径，根据已定义的特征及其相应权重产生一个优先级次序。rank.message函数依赖于很多之前定义的函数以及一个新函数get.weights，它在特征还未映射为一个权重值时执行权重查找，即主题和正文词项。

```
get.weights <- function(search.term, weight.df, term=TRUE) {
    if(length(search.term) > 0) {
        if(term) {
            term.match <- match(names(search.term), weight.df$Term)
        }
        else {
            term.match <- match(search.term, weight.df$Thread)
        }
        match.weights <- weight.df$Weight[which(!is.na(term.match))]
        if(length(match.weights) > 1) {
             return(1)
        }
        else {
```

```
            return(mean(match.weights))
        }
    }
    else {
        return(1)
    }
}
```

首先定义get.weights函数，这个函数有三个输入参数：要查找的词项（一个字符串），所要查找的对象——权重数据框，以及词项（term）的一个布尔值。最后一个参数告诉应用程序要查找的数据框是词项数据框还是线程数据框。因为thread.weights数据框的列标签存在区别，所以我们会视上述两种为完全不同的查找，需要标识这个区别。处理逻辑相当直接，因为使用了match函数在权重数据框中查找与search.term相同的元素，并返回其权重值。这里尤其需要注意函数如何处理查找失败的情况。

首先，为保险起见，检查输入get.weights的待查找词项长度是否大于0。这一步和在解析邮件数据时确保邮件真的包含主题行所做的工作是一样的类型。如果输入是一个无效的待查找词项，则简单地返回1（初等数学告诉我们，这不会影响下一步的乘积计算，因为是乘以1）。接下来，match函数会为搜索向量中所有没有匹配上search.term的元素返回NA值。因此，我们抽取那些匹配上的元素，即返回非NA值元素的权重值。如果没有一个匹配上，那么返回向量term.match的元素全是NA值，在这种情况下，match.weights的长度为0。因此，我们为这种情况增加一步检查，如果出现这种情况，还是返回1。如果匹配了若干权重值，那么就把这些权重值的均值作为结果返回。

```
rank.message <- function(path) {
    msg <- parse.email(path)
    # Weighting based on message author

    # First is just on the total frequency
    from <- ifelse(length(which(from.weight$From.EMail == msg[2])) > 0,
        from.weight$Weight[which(from.weight$From.EMail == msg[2])], 1)

    # Second is based on senders in threads, and threads themselves
    thread.from <- ifelse(length(which(senders.df$From.EMail == msg[2]))>0,
        senders.df$Weight[which(senders.df$From.EMail == msg[2])], 1)

    subj <- strsplit(tolower(msg[3]), "re: ")
    is.thread <- ifelse(subj[[1]][1] == "", TRUE, FALSE)
    if(is.thread) {
        activity <- get.weights(subj[[1]][2], thread.weights, term=FALSE)
    }
    else {
        activity <- 1
    }

    # Next, weight based on terms
    # Weight based on terms in threads
```

```
    thread.terms<-term.counts(msg[3], control=list(stopwords=stopwords()))
    thread.terms.weights <- get.weights(thread.terms, term.weights)

    # Weight based terms in all messages
    msg.terms <- term.counts(msg[4], control=list(stopwords=stopwords(),
        removePunctuation=TRUE, removeNumbers=TRUE))
    msg.weights <- get.weights(msg.terms, msg.weights)

    # Calculate rank by interacting all weights
    rank <- prod(from, thread.from, activity,
        thread.terms.weights, msg.weights)

    return(c(msg[1], msg[2], msg[3], rank))
}
```

在为从数据集中每一封邮件抽取的特征赋权重时，rank.message函数使用的规则和get.weights函数很类似。首先，它调用parse.email函数抽取我们感兴趣的四个特征。然后，用一系列的if-then语句判定抽取的特征是否出现在某个用于排序的权重数据框中，并给特征赋上相应的权重。from和thread.from利用社交特征查找发件人地址的权重。请注意，在这两种情况中，如果ifelse没匹配到任何权重，那么就返回1。这个方法和get.weights函数是相同的。

对于线程权重和词项权重，我们做了一些内部文本解析工作。对于线程特征，首先检查待排序邮件是否属于某个线程，这个过程和训练阶段没什么不同。如果属于某个线程，那么查询其活跃度排序，否则赋值为1。对基于词项的权重，我们用term.counts函数从邮件特征中获取相关词项，并获取相应的权重。在最后一步，我们把查到的所有权重值传给函数prod，产生了rank（优先级排序）值。rank.message函数返回一个含有邮件日期/时间、发件人地址、主题和优先级排序值的向量。

```
train.paths <- priority.df$Path[1:(round(nrow(priority.df) / 2))]
test.paths <- priority.df$Path[((round(nrow(priority.df) / 2)) + 1):nrow(priority.df)]

train.ranks <- lapply(train.paths, rank.message)
train.ranks.matrix <- do.call(rbind, train.ranks)
train.ranks.matrix <- cbind(train.paths, train.ranks.matrix, "TRAINING")
train.ranks.df <- data.frame(train.ranks.matrix, stringsAsFactors=FALSE)
names(train.ranks.df) <- c("Message", "Date", "From", "Subj", "Rank","Type")
train.ranks.df$Rank <- as.numeric(train.ranks.df$Rank)

priority.threshold <- median(train.ranks.df$Rank)

train.ranks.df$Priority<-ifelse(train.ranks.df$Rank >= priority.threshold, 1, 0)
```

准备启动排序算法了！在处理之前，我们要把数据按照时间拆分成两部分。第一部分就是训练数据，命名为train.paths。用这里面的数据产生排序数据，从而确定一个“优先”邮件的阈值。一旦得到了阈值，就可以在test.paths中的邮件上运行排序算法，判定哪些邮件是优先的邮件，并且对优先的邮件排定顺序。接下来，在向量train.paths

上应用rank.message函数，产生一个向量列表，其中包含每一封邮件的特征和优先级排序值。现在，我们来完成些琐碎工作，把向量列表转换成一个矩阵，最终把这个矩阵转换成带有列名的数据框以及格式正确的向量。

警告： 你可能注意到，当执行train.ranks <- lapply(train.paths, rank.message)这一行时R抛出了一个警告。这没什么问题，只不过是我们构建排序算法的方式导致的结果。如果你想关掉警告，可以在suppressWarnings函数中调用lappy。

我们现在进行关键的一步，计算优先邮件的阈值。你可以看到，在这个案例中，我们用优先级排序值的中位数作为阈值。当然，也可以用其他摘要统计量来作为这个阈值，这些都在第2章讨论过。因为并没有用到预先告知邮件应如何排序的实例来决定阈值，所以我们所进行的并不是一种标准的有监督学习任务。但是，选择中位数作为阈值有两条原则上的理由：第一，如果排序算法足够好，那么排序结果应该是一个平滑的分布，大多数邮件的优先级排序较低，少量邮件优先级排序较高。我们在寻找“重要的邮件”，也就是那些最独特的，或者说不同于电子邮件里面大众邮件的信息流。这些邮件就是排序分布中的合格部分。如果情况就是这样，那些大于中位数的值就是那些大于典型排序值的邮件。推荐优先的邮件在直观上的表现就是：挑选的邮件优先级排序值应该大于一封典型邮件的排序值。

第二，我们知道测试数据会包含这样的邮件：其特征在训练数据中完全没出现过。新邮件源源不断地流入，但是在我们当前这样的设定下，没办法更新排序算法。既如此，也许我们想设计一种倾向于包容性而非排他性的优先级规则。如果不这样做，可能会丢掉只匹配了部分特征的邮件。作为最后一步，我们给train.ranks.df加了一个Priority列，Priority是二值向量，用于指示邮件是否会作为优先邮件被排序算法推荐。

图4 5展示了在训练数据上计算的排序密度估计。垂直虚线便是中位数阈值，大约是24。你看到了吧？排序结果是很明显的重尾分布（heavy-tailed），这表明我们构建的排序算法在训练数据上表现不错。我们也看到中位数阈值非常具有包容性，大部分斜率向右下倾斜位置处的邮件都被判定为优先邮件。再强调一次，这是有意为之的。用分布的标准差作为阈值就没有这么好的包容性，我们可以用sd(train.ranks.df$Rank)计算标准差。标准差大约是90，这几乎把尾部之外所有的邮件都排除在优先邮件之外了。

现在计算测试数据集中所有邮件的优先级排序值。这个过程和计算训练数据的优先级排序值是一模一样的，为了节省篇幅就不在这里贴出代码了。要看代码，请查看本章目录下的code/priority_inbox.R文件中从第436行开始的代码。计算完测试数据的优先级排序值之后，就可以对比测试数据和训练数据，看看排序算法对新邮件排序效果如何。

图4-6在图4-5的基础上叠加了测试数据的排序密度。这个图清楚地说明了排序算法的优

劣。首先，我们注意到，测试数据的分布尾部密度更高。这意味着有更多邮件的优先级排序值不高。此外，测试数据的密度估计没有训练数据那样平滑，这说明测试数据包含了很多没出现在训练数据中的特征。因为这些特征没有在训练数据中得到匹配，于是排序算法就忽略这些特征了。

尽管仍然存在问题，但是由于我们采用了包容性较强的阈值来判定优先邮件，所以结果还不至于太糟。请注意，测试数据中处于阈值右侧的密度数量仍然比较合理。这说明我们的算法仍然能从测试数据中找到重要的邮件来推荐。最后一步，我们要检查算法到底把哪些邮件放在前面了（表4-1）。

警告： 构建这样的排序算法，本质上无法知道其好坏。在整个案例过程中，我们对每个特征都有假设，并且试图直观地进行合理解释。然而，我们永远也不可能知道这个排序算法表现到底如何，因为不可能去找这些邮件的收件人询问：排序好不好？合不合理？在分类那个案例中，因为我们知道训练数据和测试数据中每封邮件的标签，所以能通过构建混淆矩阵明确评估分类效果到底如何。在这个案例中没法这样做，但是可以通过查看结果来推断排序算法表现好坏。这也正是这个案例和标准的有监督学习的区别之一。

表4-1展示了测试数据中被排序算法标注为优先邮件的最近40封邮件。这个表格模拟的是：假如使用了这个排序算法对收件箱排序，你可能在收件箱看到的界面，并且附上了每封邮件的优先级排序值。但愿你看得惯这些有点古怪或者有争议的主题，我们会对这些结果进行分析，看看排序算法怎么对邮件分组。

这个结果最鼓舞人心的地方就是排序算法按照线程分组的效果非常好。你可以在这个表中找到这些例子，即来自同一个线程的邮件都被标记为优先邮件，而且被分到同一个组里。另外，排序算法似乎给高频发件人的邮件打了相对高的优先级分数，比如比较突出的发件人*tomwhore@slack.net*和*yyyy@spamassassin.taint.org*。最后，也许也是最鼓舞我们的，排序算法把训练数据中没出现的邮件列为优先邮件了。事实上，测试数据中被标记为优先邮件的85个线程中只有12个是训练数据的延续（约14%）。这表明我们的排序算法能把训练数据中的知识应用到测试数据的新线程中，并且无需更新就做出推荐。这太棒了！

在这一章，我们所探讨的内容从只有一种元素的特征集[译注6]拓展到有许多种特征的较复杂模型。实际上，我们完成了一个相当困难的任务，这就是在只有邮件一半事务[译注7]的前提下，设计一个邮件的排序模型。依据社交、线程活跃度以及共同词项，我们明确了四种感兴趣的特征，产生五个权重数据框来完成排序。我们刚才已经查看了结果，尽管很难明确地测试这样结果的效果如何，但是仍然很鼓舞人心。

译注6：此处指第3章仅有词项一种特征。

译注7：此外指仅有接收邮件的信息，而没有发送邮件的信息。

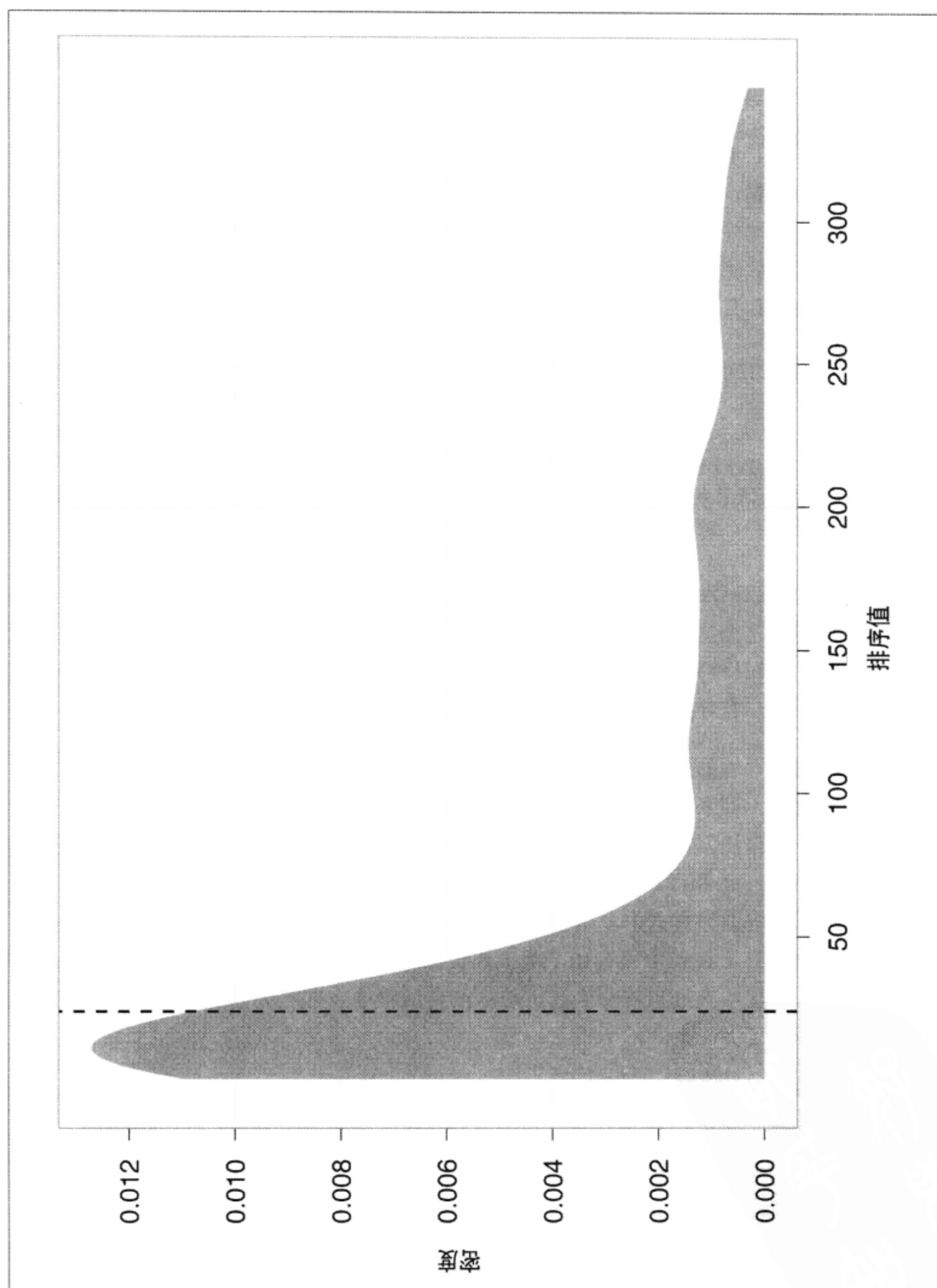


图4-5：训练数据权重密度图，并以权重的一个标准差作为优先邮件阈值

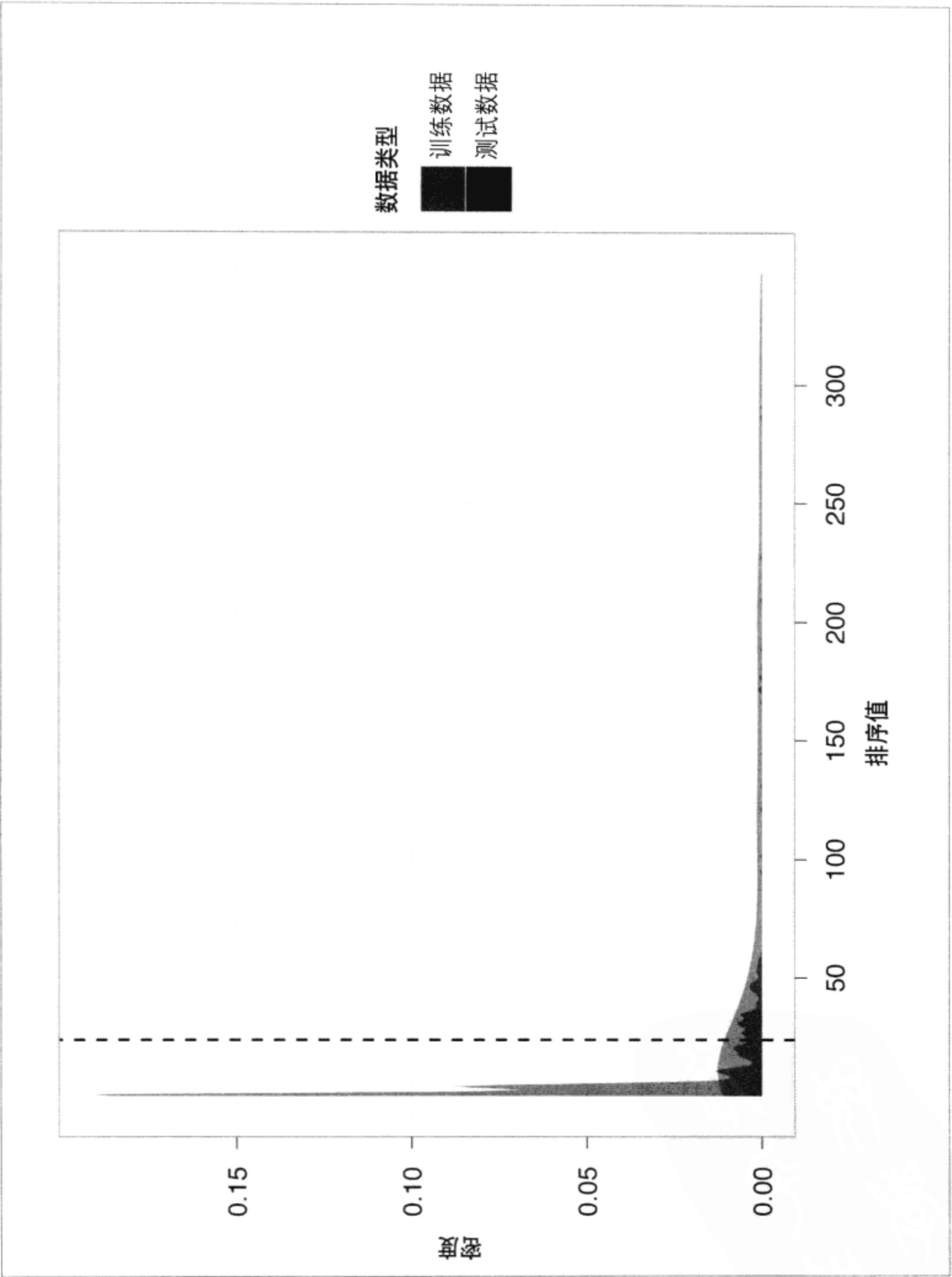


图4-6：测试数据和训练数据的权重密度图

表4-1：智能收件箱测试结果

日期	来自	主题	优先级分值
2002/12/1 21:01	geege@barrera.org	RE:Mercedes-Benz G55	31.97963566
2002/11/25 19:34	deafbox@hotmail.com	RE:Men et Toil	34.7967621
2002/10/10 13:14	yyyy@spamassassin.taint.org	RE:[SAdev]fully-public corpus of mail available	53.94872021
2002/10/9 21:47	quinlan@pathname.com	RE:[SAdev] fully-public corpus of mail available	29.48898756
2002/10/9 18:23	yyyy@spamassassin.taint.org	RE:[SAtalk] RE:fully-public corpus of mail available	44.17153847
2002/10/9 13:30	haldevore@acm.org	RE:From	25.02939914
2002/10/9 12:58	jamesr@best.com	RE:The Disappearing Alliance	26.54528998
2002/10/8 23:42	harri.haataja@cs.helsinki.fi	RE:Zoot apt/openssh & new DVD playing doc	25.01634554
2002/10/8 19:17	tomwhore@slack.net	RE:The Disappearing Alliance	56.93995821
2002/10/8 17:02	johnhall@evergo.net	RE:The Disappearing Alliance	31.50297057
2002/10/8 15:26	rah@shipwright.com	RE:The absurdities of life.	31.12476712
2002/10/8 12:18	timc@2ubh.com	[zzzzteana] Lioness adopts fifth antelope	24.22364367
2002/10/8 12:15	timc@2ubh.com	[zzzzteana] And deliver us from weevil	24.41118141
2002/10/8 12:14	timc@2ubh.com	[zzzzteana] Bashing the bishop	24.18504926
2002/10/7 21:39	geege@barrera.org	RE:The absurdities of life.	34.44120977
2002/10/7 20:18	yyyy@spamassassin.taint.org	RE:[SAtalk] RE:AWL bug in 2.42?	46.70665631
2002/10/7 16:48	owen@permafrost.net	RE:erratum [RE:no matter ...] & errors	27.09867727
2002/10/7 16:45	jamesr@best.com	RE:erratum [RE:no matter ...] & errors	27.16350661
2002/10/7 15:30	tomwhore@slack.net	RE:The absurdities of life.	47.3282386
2002/10/7 14:20	cdale@techmonkeys.net	RE:The absurdities of life.	35.11063991
2002/10/7 14:02	johnhall@evergo.net	RE:The absurdities of life.	28.16690172
2002/10/6 17:29	geege@barrera.org	RE:Our friends the Palestinians, Our servants in government.	28.05735369
2002/10/6 15:24	geege@barrera.org	RE:Our friends the Palestinians, Our servants in government.	27.32604275
2002/10/6 14:02	johnhall@evergo.net	RE:Our friends the Palestinians, Our servants in government.	27.08788823
2002/10/6 12:22	johnhall@evergo.net	RE:Our friends the Palestinians, Our servants in government.	26.48367996
2002/10/6 10:20	owen@permafrost.net	RE:Our friends the Palestinians, Our servants in government.	26.77329071

表4-1：智能收件箱测试结果（续）

日期	来自	主题	优先级分值
2002/10/6 10:02	fork_list@hotmail.com	RE:Our friends the Palestinians, Our servants in government.	29.60084489
2002/10/6 0:34	geege@barrera.org	RE:Our friends the Palestinians, Our servants in government.	29.35353465

在第3章和本章中，你已经经历了相当简单的有监督学习案例，即垃圾分类以及一个非常基本的启发式排序形式。现在你要准备好见识统计学习中的中流砥柱：回归。

第5章

回归模型：预测网页访问量

回归模型简介

从理论上讲，回归模型是一个非常简单的概念，即用已知的数据集来预测另外一个数据集。例如，保险精算师也许想在已知人们吸烟习惯的基础上预测其寿命，气象学家也许想使用已知过去每天的温度来预测明天的温度。一般来说，我们称已知的数据为输入，将你想要预测的数据称为输出。有时候，人们也会把输入叫做预测变量或者特征。

回归模型和分类模型的不同之处在于，回归模型的输出是真正的数字。比如我们在第3章中描述的那些分类问题中，用于区分类型的虚拟变量必须是数字，因此0代表合法邮件，而1代表垃圾邮件。但是这些数字仅仅只是符号，当我们使用虚拟变量的时候，并不去研究0或者1的数字性质。在回归模型中，其输出本质的事实就是它的输出是真正的数字，例如，预测温度这类目标的范围也许会在50°~70°。因为预测结果是数字，所以可以得出输入和输出之间关系的有力断言，例如，你也许能得出这样的结论，当人们每天所抽香烟的包数增加一倍时，他们的预计寿命将会减少一半。

当然，这里存在的问题是，想要做出精确的数值预测和能够实际做出这样的预测并不是一回事。为了能够做出定量的预测，需要提出一些规则，这些规则能够利用我们可用的信息。近200年来，统计学家们提出了各种各样的回归算法，这些算法可以用不同的方式根据输入预测输出。在本章中，我们会讲到最常用的回归模型：线性回归（linear regression）。

基准模型

下面要说的也许有点愚蠢，但是不妨一听：对于已有输入信息的使用，可能最简单的方

法就是完全忽略所有输入，然后将计算过去所观测的输出值的均值作为预测结果。在保险精算师的例子中，我们可以完全忽略一个人的健康记录并且预测其寿命等于人类平均寿命。

对于每种情况都将平均值作为结果并不像看上去那样幼稚：如果我们要在不使用其他任何信息的情况下，尽可能做出接近事实的预测，那么将平均输出作为结果是我们可以做出的最好预测。

注意： 这里没有定义“最好”的含义。说了半天“最好”，却没有明确到底什么是“最好”，如果这样惹你不高兴了，那么我们保证稍后会给出一个正式的定义。

在我们讨论如何做出最好的合理预测之前，假定我们拥有一份数据集，它来源于一个虚构的保险统计数据库，其中一部分如表5-1所示。

表5-1：关于寿命的保险统计数据

是否吸烟	死亡年龄
1	75
1	72
1	66
1	74
1	69
…	…
0	66
0	80
0	84
0	63
0	79
…	…

面对这个全新的数据集，一个好的想法是根据第2章中的建议，在做任何正式分析之前，先对数据进行一些分析。我们有一列数据和另外一列虚拟的编码因子，因此我们很自然地想到画出密度图来比较吸烟者和非吸烟者（画图结果见图5-1）。

```
ages <- read.csv('data/longevity.csv')

ggplot(ages, aes(x = AgeAtDeath, fill = factor(Smokes))) +
geom_density() +
facet_grid(Smokes ~ .)
```

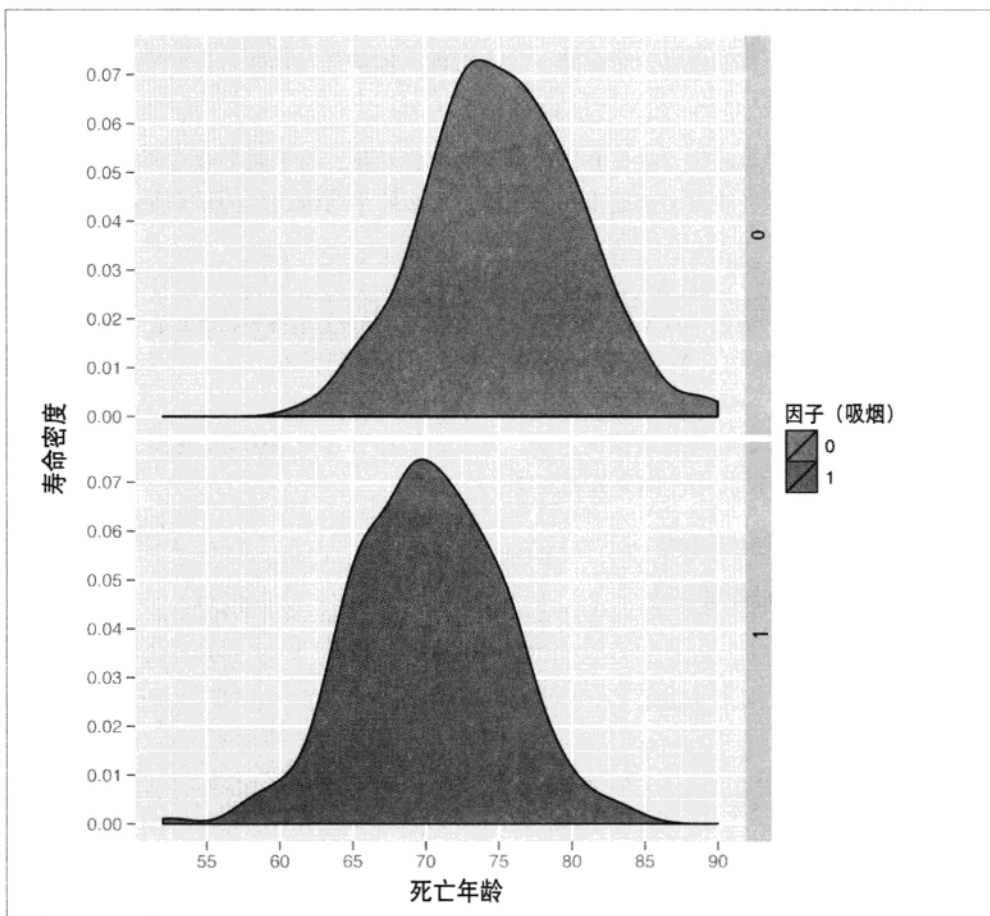


图5-1：1000人的寿命密度图，以是否吸烟为因子

从这个寿命密度图可以看出，我们有理由相信吸烟习惯和寿命有关系，因为不吸烟的人寿命分布中心和吸烟的人相比，向右偏移。换句话说，不吸烟的人平均寿命比吸烟的人平均寿命长。关于你可以怎样使用人们有关吸烟习惯的信息来预测其寿命，我们会给出描述，但是在这之前，暂时假装并没有任何这方面的信息。在这种情况下，无论你要研究的新人有什么样的吸烟习惯，你都只需选择一个单独的数字作为你对其寿命的预测。那么你应该选择什么样的数字？这个问题并不简单，因为选择哪个数字，取决于你对“好的预测”的评判标准。有许多合理的方法来定义预测的准确性，但是实际上有一个衡量指标在整个统计学的历史中占有统治地位。这个衡量指标叫做平方误差。如果你想要预测的是y值（真实结果）并且你的假设是h（你关于y值的假设），那么假设的平方误差计算很简单，就是（y-h）^2。

之所以要用平方误差来衡量预测的好坏，其中除了有遵循惯例，从而让其他人能够理解的固有原因外，还有很多其他的原因。我们现在不去深究这些原因，但是在第7章讨论机器学习中的优化算法时，对于人们用来度量误差的各种方法，将讨论这个问题稍微更多一点。目前，我们仅试图让你确信一些基本的东西：如果使用平方误差作为预测质量的衡量指标，那么对人的寿命做出的最好假设（在没有任何关于人的习惯的附加信息的情况下）就是人的寿命均值。

为了使你相信这个结论，让我们看看如果使用其他假设来代替均值的话会发生什么。使用上面的寿命数据集，`AgeAtDeath`（死亡年龄）的均值是72.723，暂时将这个数向上取整为73。接着我们可以提出这样的问题，“如果假设每个人都活到73岁，那么我们对于上述数据集中的人们的年龄所做出的这个预测到底有多差？”

可以使用R语言来回答这个问题，下面的这段代码对数据集中关于每个人的平方误差进行求和，由此计算出平方误差的均值，我们把平方误差的这个均值叫做均方误差（Mean Squared Errer，MSE）。

```
ages <- read.csv('data/longevity.csv')

guess <- 73

with(ages, mean((AgeAtDeath - guess) ^ 2))
#[1] 32.991
```

运行上述代码之后我们发现，通过假设已有数据集中每个人的寿命都是73岁而得到的均方误差是32.991。但是，仅通过这个数字本身，并不足以使你相信，我们使用除73之外的数字作为假设会得到更差的结果。为了让你相信73是最好的假设，我们需要考虑如果使用某些其他的可能假设，得到的结果如何？为了使用R语言来计算这些其他的结果，我们在范围为63～83的可能假设序列上做一个循环。

```
ages <- read.csv('data/longevity.csv')

guess.accuracy <- data.frame()

for (guess in seq(63, 83, by = 1))
{
  prediction.error <- with(ages,
                           mean((AgeAtDeath - guess) ^ 2))
  guess.accuracy <- rbind(guess.accuracy,
                          data.frame(Guess = guess,
                                     Error = prediction.error))
  }
  ggplot(guess.accuracy, aes(x = Guess, y = Error)) +
    geom_point() +
    geom_line()
```

如图5-2所示，使用除73之外的其他任何假设对于我们的数据集来说带来的都是更差的预测。这实际上是一个我们可以从数学上证明的一般理论结果：为了最小化均方误差，你需要使用你的数据集的均值作为预测。这说明了很重要的一点：在已有了关于吸烟信息的情况下做出预测，如果要衡量其好坏，那就应该看它比你对每个人都用均值去猜的结果提升了多少。

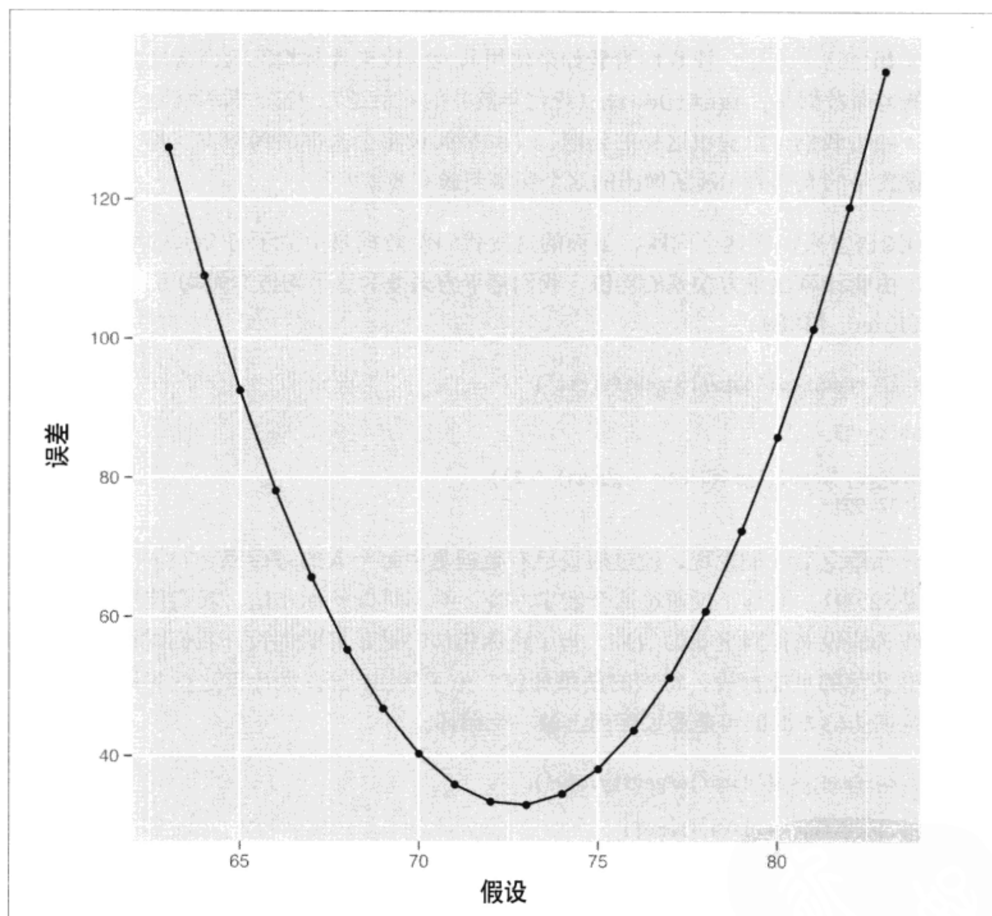


图5-2：均方误差（MSE）

使用虚拟变量的回归模型

那么如何使用那些信息呢？在处理更大的问题之前，让我们从一个简单的例子开始。如何使用是否吸烟这样的信息来对人的寿命做出更好的假设？

一个简单的想法是，先分别估计吸烟的人和不吸烟的人的死亡年龄均值，然后根据你

要研究的人是否吸烟，以对应均值作为其预测寿命。这一次，我们将使用均方根误差（Root Mean Squared Error，RMSE）来代替均方误差（MSE），它在机器学习文献中更加常见。

下面是将吸烟的人和不吸烟的人分成单独建模的两组之后，使用R语言计算均方根误差（RMSE）的一种方法（结果见表5-2）。

```
ages <- read.csv('data/longevity.csv')

constant.guess <- with(ages, mean(AgeAtDeath))
with(ages, sqrt(mean((AgeAtDeath - constant.guess) ^ 2)))

smokers.guess <- with(subset(ages, Smokes == 1),
                      mean(AgeAtDeath))
non.smokers.guess <- with(subset(ages, Smokes == 0),
                          mean(AgeAtDeath))
ages <- transform(ages,
                  NewPrediction = ifelse(Smokes == 0,
                                         non.smokers.guess,
                                         smokers.guess))
with(ages, sqrt(mean((AgeAtDeath - NewPrediction) ^ 2)))
```

表5-2：使用了更多信息之后的预测误差

信息	平均平方根误差（RMSE）
不包含吸烟信息的预测误差	5.737096
包含吸烟信息的预测误差	5.148622

通过观察得到的均方根误差（RMSE），可以看到，在给我们研究的人群引入了更多信息之后，所做出的预测确实更好了：当引入关于吸烟习惯的信息之后，在预测人群寿命时的预测误差减少了10%。一般来说，每当我们有了可以将数据点分成两种类型的二元区分性质——假设这些二元区分性和我们尝试预测的结果相关，我们都能得到比仅仅使用均值更好的预测结果。简单二元区分性的例子有：男人和女人，美国政治中的民主党人和共和党人。

因此我们现在有了将虚拟变量整合进预测的机制。但是数据中那些关于预测对象的更丰富的信息，该如何使用呢？关于“更加丰富”，我们的意思是指两点：第一点，我们想要知道如何使用那些不是二元区分性的输入，如身高和体重这样的连续型数值；第二点，我们想要知道如何一次使用多重信息源以提升预测。在前面的保险精算师例子中，假设已知：a）某个人是否吸烟；b）他父母的死亡年龄。我们的直觉是，拥有这两个不同的信息源，应该比使用单个信息源能够揭示更多东西。

但是，把已有的所有信息都用起来，并不是一个简单的事情。在实践中，我们需要做出某些简化问题的假设以使得问题圆满解决。我们将要描述的假设是那些作为线性回归模

型的基础的假设，线性回归模型是我们在本章中唯一描述的一种回归模型。仅仅使用线性回归模型，实际上比可能看上去要受到更少的约束，因为回归模型的实际应用中90%使用的都是线性回归模型，并且只需要花很少的工作量就可以将其修改成形式更加复杂精练的回归模型。

线性回归简介

当使用线性回归模型预测输出结果时，所做的最大的两个假设如下。

可分性/可加性

如果有多份信息可能影响我们的假设，那么通过累加每一份信息的影响来产生我们的假设，就像单独使用每份信息时一样。例如，如果酗酒者比不酗酒者少活1年，并且吸烟者比不吸烟者少活5年，那么一个吸烟的酗酒者应该会比既不吸烟也不酗酒的人少活6(1+5)年。这种假设在事情同时发生时将它们的单独影响累加到一起，是一个很大的假设，但是这是很多回归模型应用的不错起点。后面将讲解相互作用的概念，这便是线性回归模型的可分性，比如说，你知道如果你吸烟，那么酗酒就会让情况变得更糟糕。

单调性/线性

当改变一个输入值总是使得预测的输出结果增加或者减少时，这个模型是单调的。例如，如果你使用身高作为输入值预测体重，并且模型是单调的，那么当前的预测是每当某些人的身高增加，他们的体重将会增加。单调性是个很强的假设，因为你可以想出很多输出结果随着输入增大而增加一点点，然后突然开始减少的例子，但是跟线性回归算法中的线性假设相比，单调性还不算一个特别强的假设。线性是一个具有非常简单意义的技术术语：如果你在一个散点图中画出输入和输出结果，你应该会看到一条将输入和输出结果联系起来的直线，而不是某种更复杂的形状，如曲线或者波浪线。对于不习惯形象思维的人来说可以这样来比喻，线性意味着每当改变一个单位的输入，输出结果总是增加N个单位或者输出结果总是减少N个单位。每一个线性模型都是单调的，但是曲线可以在非线性的情况下也是单调的。基于这个原因，线性比单调性具有更强的约束性。

注意： 通过观察图5-3，我们可以清楚地明白此前所说的直线、曲线和波浪线是什么意思。曲线和直线都是单调的，但是波浪线是非单调的，因为它时而上升时而下降。

仅当数据的输入输出关系图看起来像是直线时才叫做标准线性回归。实际上可能也会用线性回归去拟合曲线或波浪线，但那是第6章才会讲到的进阶主题。

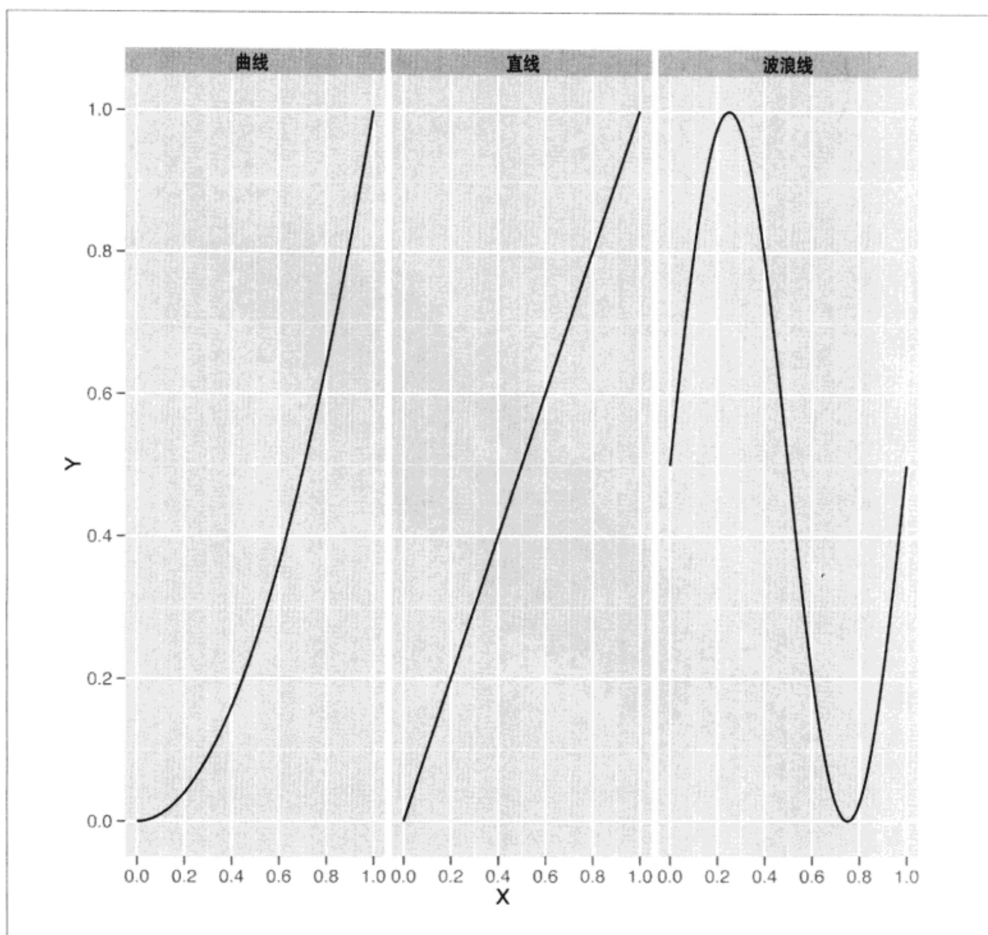


图5-3：曲线、直线和波浪线

将可加性假设和线性假设牢记在心中，让我们开始完成一个简单的线性回归模型的例子。再一次回到前面案例中的身高和体重数据集上，以此展示如何在这个情境中使用线性回归模型。在图5-4中，可以看到之前已经画过多次的散点图。对绘图代码稍作改动，我们就能看到线性回归模型生成的那条直线，可以凭这条直线来用身高预测体重，如图5-5所示。只需在调用geom_smooth函数时指明要用lm方法即可，其中lm方法已经实现了“线性模型”。

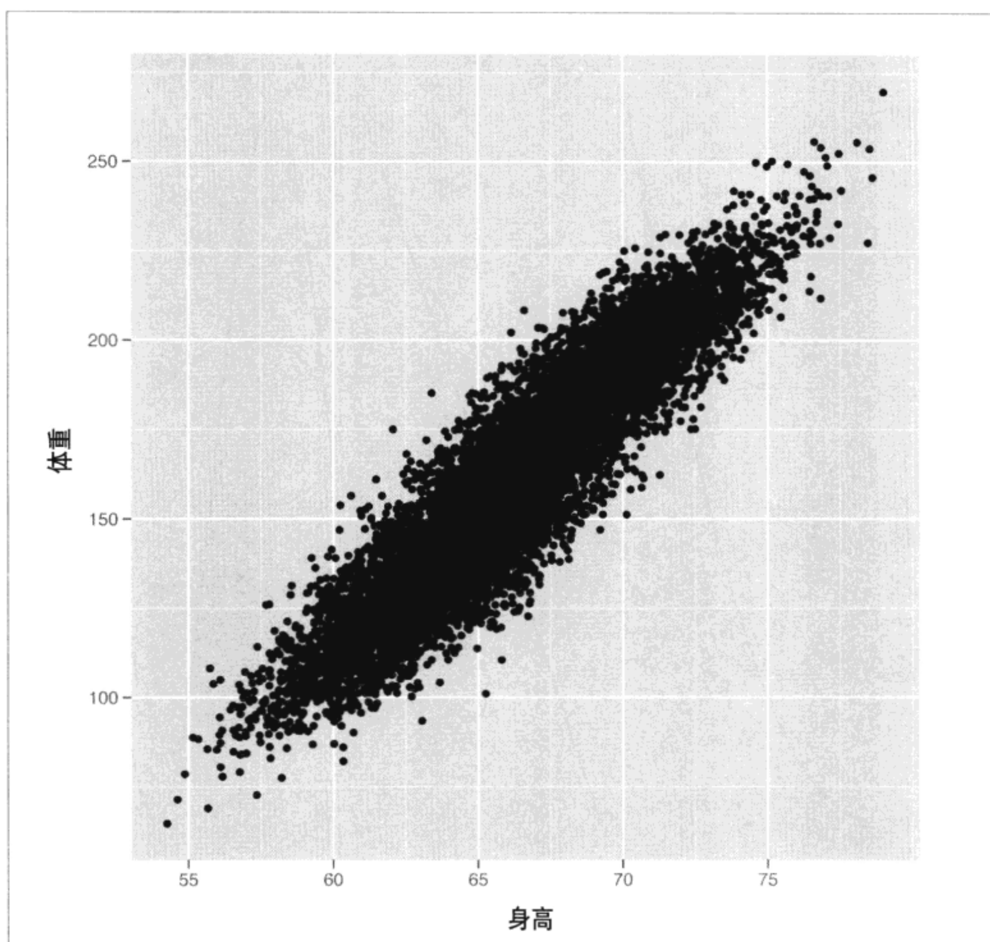


图5-4：体重相对身高的散点图

```
library('ggplot2')

heights.weights <- read.csv('data/01_heights_weights_genders.csv',
                            header = TRUE,
                            sep = ',')

ggplot(heights.weights, aes(x = Height, y = Weight)) +
  geom_point() +
  geom_smooth(method = 'lm')
```

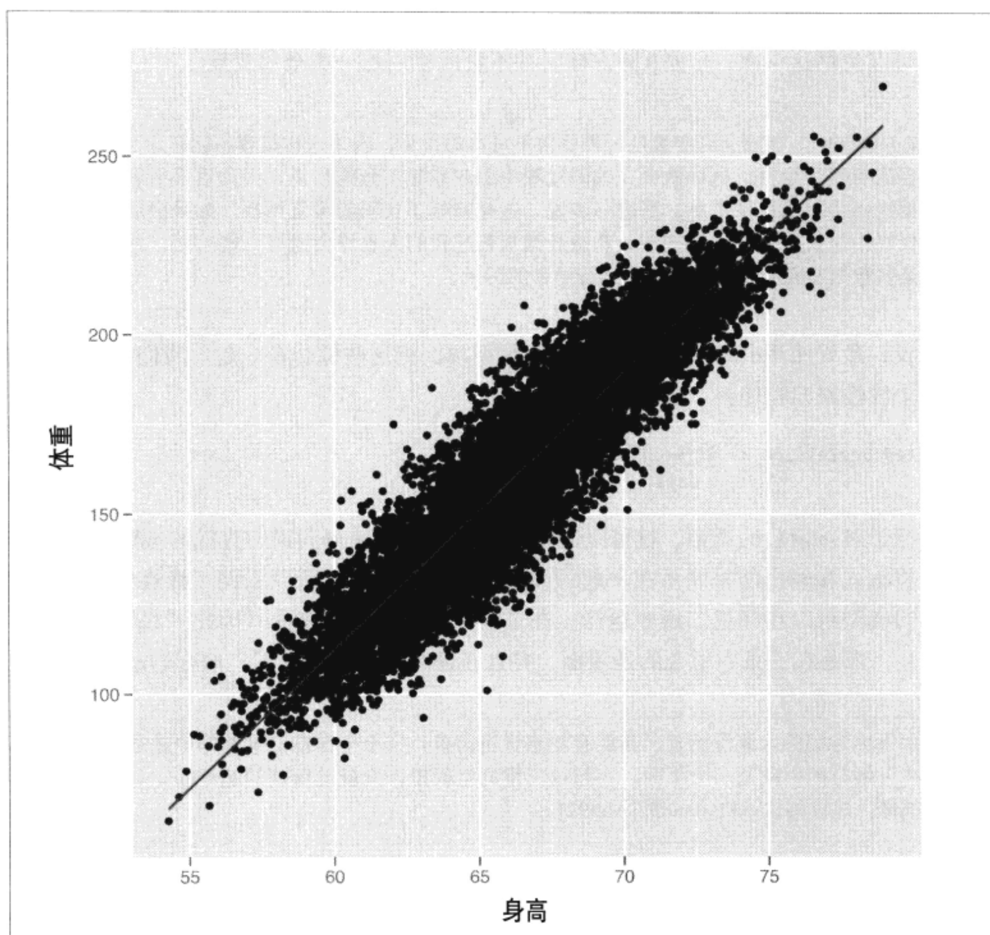


图5-5：增加了回归直线后体重相对身高的散点图

从图5-5应该可以看出，通过这条直线，在已知一个人身高的前提下去预测其体重会取得非常好的效果。例如，我们看着这条直线会说，应该预测某个身高是60英寸的人的体重为105磅，并且应该预测某个身高为75英寸的人体重为225磅。因此，使用一条直线来做出预测是一个明智的选择，这一结论看上去有理由被人们接受。于是，我们需要回答的问题变成了："如何找到用于定义在这幅图中看到的直线的数字？"这正是R语言作为一门机器学习语言真正擅长的地方：R语言中一个称为`lm`的简单函数将会为我们完成所有这些工作。为了使用`lm`，我们需要使用~操作符指明一个回归公式。在这个例子中，需要从身高来预测体重，因此我们的公式写作`Weight~Hight`。如果我们将要做出的预测是相反的方向，应该将公式写成 `Hight~Weight`。如果你要问这些公式怎么读，我个人喜欢把`Hight~Weight`读作"身高关于体重的函数"。除了关于回归模型的详细描述之外，

我们还需要告诉R语言使用的数据存放在哪里。如果你的回归模型中的变量都是全局变量，你可以忽略这一点，但是R语言社区并不赞成使用那一类全局变量。

警告： 在R语言中，存放全局变量是一种非常不可取的行为，因为这样很容易就忘记了被加载到内存的有哪些数据。这样做会在编码过程中引起意想不到的后果，因为这样很容易失去对数据进出内存的记录信息。更进一步说，这样降低了代码的可复用性。如果全局变量没有在代码中显式地定义，某些不太熟悉数据加载过程的人也许会遗漏一些东西，并且由于缺少某些数据集或变量而产生代码运行异常的后果。

对你来说，最好使用data参数显式地指定数据源。把这些综合在一起，我们用如下方式运行一个线性回归程序：

```
fitted.regression <- lm(Weight ~ Height,
                        data = heights.weights)
```

一旦运行了对lm函数的调用，就可以通过调用coef函数来得到回归直线的截距，coef函数返回将输入和输出结果联系在一起的线性模型的系数。之所以强调“线性模型”，是因为回归模型可以在超过二维中运行。在二维中，拟合的是直线（因此“线性”就是指这部分），但是在三维中拟合的是平面，并且在超过三维的情况下，拟合的是超平面。

注意： 如果你对那些术语没感觉，其实直观感觉也简单：一个平面在二维空间中就是一条直线，在三维空间中就是一个平面，在超过三维的空间里，它就是所谓的超平面。如果你还不是很懂，建议阅读《Flatland》[Abb92]。

```
coef(fitted.regression)
#(Intercept) Height
#-350.737192 7.717288
```

从某种意义上来说，理解了这个输出就意味着理解了线性回归模型。想要了解coef输出的意义，最好的办法是显式地把输出所隐含的关系写出来：

```
intercept <- coef(fitted.regression)[1]
slope <- coef(fitted.regression)[2]

# predicted.weight <- intercept + slope * observed.height
# predicted.weight == -350.737192 + 7.717288 * observed.height
```

这也就是说，每当某个人的身高增加一英寸，就会导致他的体重增加7.7镑。这样的关系让我们觉得相当合理。相比之下，截距就有点奇怪了，因为它告诉你一个身高为0英寸的人应该有多重。根据R语言程序的结果，这样的人应该重350镑。如果你做一些代数运算，就能推断出在这个预测算法中，一个人至少要有45英寸身高，才能显示出其体重0镑。简言之，我们的回归模型对于儿童或者身高特别矮的成年人来说并不是太适用。

这实际上是线性回归模型的一般性系统问题：当输入数据偏离既有输入观测数据时，预测模型一般不擅长预测其输出结果[注1]。通常，你可以做一些工作改善对那些用于训练模型的数据范围之外的数据所做的假设的质量。但是在这个例子中也许没有这个必要，因为人的体重通常在48~96英寸。

除了仅仅运行回归模型并且观察系数结果，还可以使用R语言做更多的事情，用以理解线性回归的结果。由于介绍R语言所能产生的所有不同输出将会占用整本书的篇幅，因此这里仅仅关注提取系数之后的一些最为关键的部分。当在实际环境中进行预测的时候，系数就是你所需要知道的全部。

首先，需要对模型存在什么问题有一个感性的认识。我们通过计算模型的预测结果，并且将结果与输入做比较来达到上述目的。为了获得模型的预测结果，可以使用R语言中的predict函数：

```
predict(fitted.regression)
```

一旦获得了模型的预测结果，就可以使用简单的减法来计算预测结果和真实值之间的误差。

```
true.values <- with(heights.weights,Weight)
errors <- true.values - predict(fitted.regression)
```

通过上述这种方法计算得出的误差称为残差，因为它们是我们的数据中由一条直线所能解释的那部分之外的剩余部分。在R语言中可以通过使用residuals函数替换predict函数来直接获得残差：

```
residuals(fitted.regression)
```

为了发现使用线性回归时产生的明显错误，可以把残差和真实数据对应画在一幅图中。

```
plot(fitted.regression, which = 1)
```

注意： 在这里，通过指定which=1，仅让R语言只画出了第一个回归诊断点图。你也可以获得其他的点图，建议你通过实践看看其他的点图对你是否有所帮助。

在这个例子中，我们可以说这个线性模型很有效，因为在残差中不存在系统性的结构。但是这里有一个直线并不适用的例子：

```
x <- 1:10
y <- x ^ 2
```

注1：用技术方式描述这一问题就是：回归擅长内推插值（interpolation），却不擅长外推归纳（extrapolation）。

```
fitted.regression <- lm(y ~ x)
plot(fitted.regression, which = 1)
```

对于这个问题，我们可以看到残差中存在明显的结构。当对数据进行建模时，通常碰到的一件难办的事情是：一个模型应该把真实世界的信号（预测值给出的）和噪声（残差给出的）分开来。如果你通过肉眼就能看到残差中存在的信号，那么你的模型就没有强大到足以提取所有的信号，并且只留下真正的噪声作为残差。为了解决这个问题，第6章将讨论比我们现在使用的简单线性回归模型更强大的回归模型。但是能力越大，责任就越大，第6章实际上将专注于因使用那些强大的模型而出现的特殊问题，那些模型实在是太强大了，以致不小心谨慎就不能使用。

接下来我们将更仔细地讨论正在使用的线性回归模型做得有多好。拥有残差是很棒的事情，但是在使用时，不可抗拒地要处理太多误差。我们想通过单独的一个数字来总结结果的质量。

最简单的误差衡量指标是：1）取所有的残差；2）对它们进行平方处理，以获取模型的误差平方；3）把这些误差平方加在一起求和。

```
x <- 1:10
y <- x ^ 2

fitted.regression <- lm(y ~ x)

errors <- residuals(fitted.regression)
squared.errors <- errors ^ 2
sum(squared.errors)
#[1] 528
```

对于比较不同的模型，这个简单的误差平方和数值是有用的，但是它存在一些缺点，致使大多数人最终讨厌它。

首先误差平方和在大数据集上的值比在小数据集上的值更大。为了确信这一点，想象一下已有一个固定的数据集以及这个数据集的误差平方和，现在增加一个不能完美预测的数据点。这个新数据点将误差平方和增加，这是因为添加一个正数到之前的误差平方和只能使得这个和更大。

但是有一个简单的方法可以解决这个问题：使用误差平方的均值，而不是用它们的和。这也就是本章前面已使用过的均方误差（MSE）度量方法。尽管MSE并不会在我们获取更多数据时像误差平方和那样一直增加，但是MSE依然存在一个问题：如果预测的平均偏离量只有5，那么均方误差数将会是25。这是因为我们对误差求了平方，然后再计算它们的均值。

```
x <- 1:10
```

```
y <- x ^ 2

fitted.regression <- lm(y ~ x)

errors <- residuals(fitted.regression)
squared.errors <- errors ^ 2
mse <- mean(squared.errors)
mse
#[1] 52.8
```

这个关于度量问题的解决方法很简单：对均方误差（MSE）进行开方运算以获得均方根误差，这就是RMSE度量方法，在本章的前面曾经尝试过。RMSE是一个非常流行的度量方法，一般用于评估机器学习算法的效果，包括那些比线性回归更加复杂精巧的算法。例如，Netflix大奖赛就是使用RMSE作为明确的标准来进行评分的，它用RMSE来评价参赛者的算法效果有多好。

```
x <- 1:10
y <- x ^ 2

fitted.regression <- lm(y ~ x)

errors <- residuals(fitted.regression)
squared.errors <- errors ^ 2
mse <- mean(squared.errors)
rmse <- sqrt(mse)
rmse
#[1] 7.266361
```

RMSE有一点不尽如人意，就是它不能让人直观清楚地看出哪个模型表现平平。理想的效果很清晰，就是RMSE值为0，但在实际任务中，追求理想是不切实际的。同样，使用RMSE也不容易识别什么时候一个模型的效果非常差。例如，如果每个人的身高都是5英尺，而你的预测是5000英尺，你将得到一个巨大的RMSE值。并且你可以做得比那个预测更差，比如预测出是50 000英尺，再差一些就是5 000 000英尺。RMSE可以取无限的值，这使得通过它很难知道你的模型效果是否合理。

当我们使用线性回归模型时，解决这个问题的办法是使用R^2。R^2的思路是要看你的模型与假如只是用均值作为预测结果相比有多好。为了便于解释，R^2的值将总是介于0～1。如果你正在做的预测并不比使用均值做得更好，R^2的值将是0。而如果你对于每个数据点都做出完美的预测，R^2的值将是1。

因为R^2的值总是在0～1，所以人们倾向于将它乘以100，并且将相乘的结果看做数据中能够被你的模型解释的那部分所占的百分比。这是一种对模型准确性建立直观认识的简便方法，甚至在一个新领域，即使你没有任何关于RMSE的经验标准值，都可以使用它。

要计算R^2，需要计算两个RMSE值，第一步是只使用均值来当做所有样本数据的预测值时的RMSE，第二步是使用你的模型所做出的预测的RMSE。完成这两步之后，只需要一个简单的除法算术运算就能得到R^2，代码描述如下所示：

```
mean.rmse <- 1.09209343
model.rmse <- 0.954544

r2 <- 1 - (model.rmse / mean.rmse)
r2
#[1] 0.1259502
```

预测网页流量

到目前为止，我们已经做好了使用回归模型开展工作的准备，本章的案例研究将专注于使用回归模型预测互联网上排名前1000的网站在2011年的访问量。数据集（由Neil Kodner提供）的前五行见表5-3。

我们只需要用到这个数据集的一部分列来开展工作，只考虑如下五个列：Rank、PageViews、UniqueVisitors、HasAdvertising和IsEnglish。

Rank列表明这个网站在前1 000的排名列表中的位置。正如你所看到的，Facebook是数据集中排名第一的网站，而YouTube是排名第二的网站。Rank是一类有意思的度量标准，因为它是一个顺序值，所使用的数字并不是真的要表示一个值，而仅仅用于表明网站流量的排名。针对这些值无意义，你可以这样理解：像“这个列表中排在第1.578位的网站是什么？”这类问题是没有真正答案的。如果所使用的数字是基数值（cardinal value），这类问题将会有答案。还可以通过另一种方式来强调这种区别，你可以注意到，如果用A、B、C、D来代替1、2、3、4表示排名，这种方式并不会丢失任何信息。

表5-3：排名最高网站的数据集合

Rank	Site	Category	Unique-Visitors	Reach	Page-Views	HasAd-vertising	InEnglish	TLD
1	Facebook.com	Social Networks	880000000	47.2	9.1e+11	Yes	Yes	com
2	youtube.com	Online Video	800000000	42.7	1.0e+11	Yes	Yes	Com
3	yahoo.com	Web Portals	660000000	35.3	7.7e+10	Yes	Yes	Com
4	live.com	Search Engines	550000000	29.3	3.6e+10	Yes	Yes	Com
5	Wikipedia.org	Dictionaries & Encyclopedias	490000000	26.2	7.0e+09	no	Yes	org

接下来的列是PageViews，这是我们在这个案例研究中想要预测的输出，它告诉我们一个网站在那一年被访问了多少次。这是一个衡量网站流行程度的好方法，比如Facebook拥有重复访问的用户，这些用户多次回访该网站。

注意： 在你读完本章之后，尝试比较PageViews和UniqueVisitors是一个不错的练习，从中可以发现一种方法，该方法能指出哪类网站会有很多的重复访问者，而哪类网站有非常少的重复访问者。

UniqueVisitors这一列告诉我们，在采样的月份期间，有多少不同的用户访问网站。如果你认为PageViews容易因为用户不必要的刷新页面而增加，那么用UniqueVisitors这列来衡量有多少不同的用户访问网站是一个不错的方法。

HasAdvertising这一列告诉我们一个网站上面是否有广告。你也许会认为广告是噪声，并且在其他所有条件相同的时候，人们倾向于不访问那些有广告的网站。我们可以使用回归模型做个显而易见的测试。事实上，回归模型最大的一个价值就是它让我们能尝试回答一些问题，在这些问题中我们不得不讨论“其他所有条件相同”的情况。之所以这里说“尝试”，是因为回归模型的质量，只与我们拥有的输入的好坏相一致。如果有一个重要的变量从我们的输入中丢失了，回归模型的结果就可能与真相相差非常远。因此，你应该一直假设回归模型的结果是不确定的：“如果我们拥有的输入足够回答这个问题，那么答案应该是……”

IsEnglish这一列告诉我们一个网站的主要语言是否是英语。审视整个列表，可以很清晰地看到，大多数排名靠前的网站主要语言不是英语就是汉语。我们之所以选择这一列，是因为有一个有趣的问题：用英语作为网站主要语言是一个正面因素还是负面因素？选择它的另一个原因是：这个回归例子的因果关系并不是完全清楚的，因为用了英语，所以网站才会广受欢迎？还是因为英语是互联网的通用语言，所以广受欢迎的网站决定采用英语？回归模型可以告诉你这两件事之间有关系，但是它不能告诉你的是，其中哪一件是因，哪一件是果。

现在我们已经描述了已有的输入，并且选定PageViews作为输出。首先从直观上认识一下这些东西之间的关联。一开始，我们将绘制一幅PageViews和UniqueVisitors关联的散点图。我们一直建议读者在用回归模型关联数值型变量之前绘制它们的散点图，因为可以从散点图上清晰地看出回归模型的线性假设是否成立。

```
top.1000.sites <- read.csv('data/top_1000_sites.tsv',
                           sep = '\t',
                           stringsAsFactors = FALSE)
ggplot(top.1000.sites, aes(x = PageViews, y = UniqueVisitors)) +
  geom_point()
```

我们通过调用ggplot函数获得的散点图（如图5-6所示）看上去很糟糕：几乎所有的数值都在x轴的附近挤成一束，而只有非常少的数字跳出了那一堆数。这是使用非标准分布数据工作时常见的一个问题，为显示整个数值跨度而选择使用非常大的刻度，使得数据中主要数据点趋向于彼此之间距离很近，以致无法直观地将它们区分开。为了证实数据的散点图上的那个糟糕形状，是因为使用了这种大刻度画图方法的问题，可以观察PageViews本身的分布：

```
ggplot(top.1000.sites, aes(x = PageViews)) +
  geom_density()
```

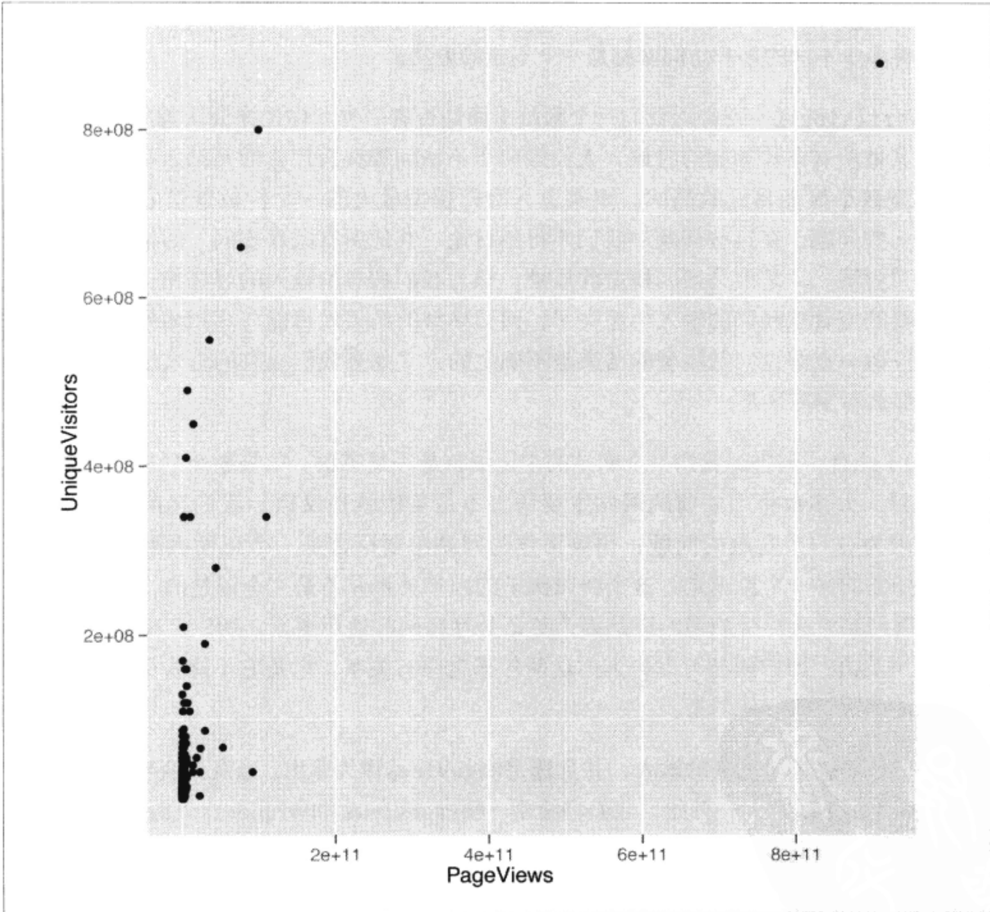


图5-6：UniqueVisitors对PageViews的散点图

这个密度图（如图5-7所示）看上去和前面的散点图一样完全不可理解。当你看到没有意

义的密度图时，一个好办法是尝试对你想要分析的数值取log，并且使用经过log变换后的值重绘一幅密度图。我们可以简单地通过调用R语言的log函数来实现上述想法：

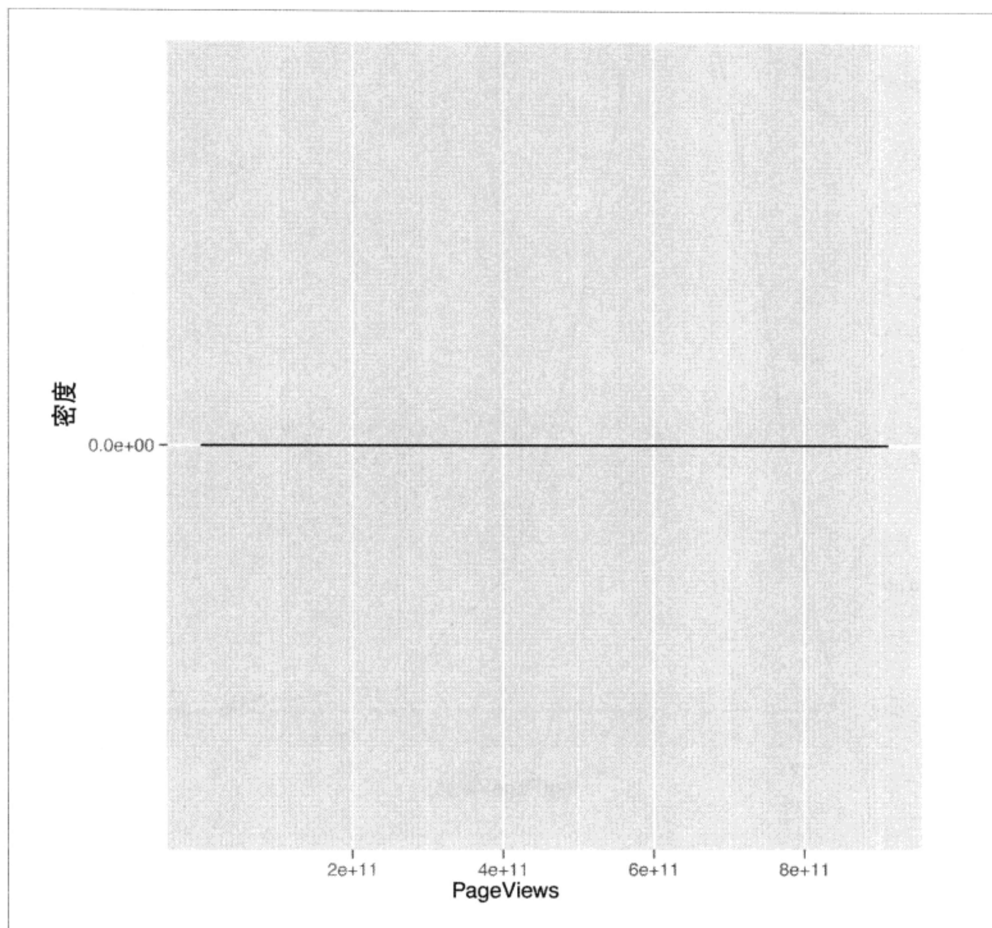


图5-7：PageViews的密度图

```
ggplot(top.1000.sites, aes(x = log(PageViews))) +
geom_density()
```

这样画出的密度图（如图5-8所示）看上去就合理多了。因此，从现在开始我们将开始使用log变换后的PageViews和UniqueVisitors。很容易在log刻度上重新绘制前面那样的散点图：

```
ggplot(top.1000.sites, aes(x = log(PageViews), y = log(UniqueVisitors))) +
geom_point()
```

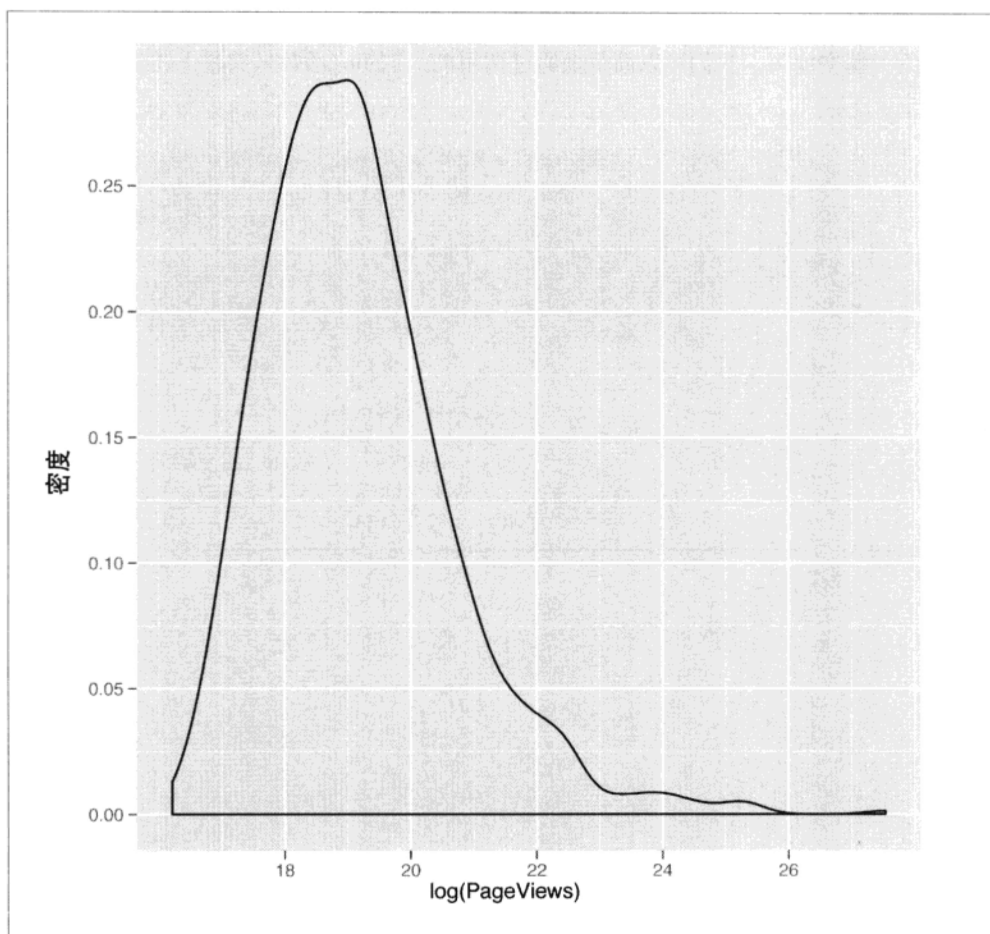


图5-8：关于Pageviews的log刻度密度图

ggplot2程序包内置了一个方便的函数，用于把坐标轴的刻度转换成log形式。在这个案例中，可以使用scale_x_log或者scale_y_log。不过，回忆一下第4章所讨论的内容，在某些情况下，你可能想用logp函数来避免对0取log的错误。可是，在这个例子中，不存在那样的问题。

散点图的作图结果如图5-9所示，看上去好像有一条可以使用回归模型画出的潜在的直线。在使用lm函数去拟合一条回归直线之前，让我们以method='lm'为参数使用geom_smooth函数来看看回归直线将是什么样子的：

```
ggplot(top.1000.sites, aes(x = log(PageViews), y = log(UniqueVisitors))) +
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE)
```

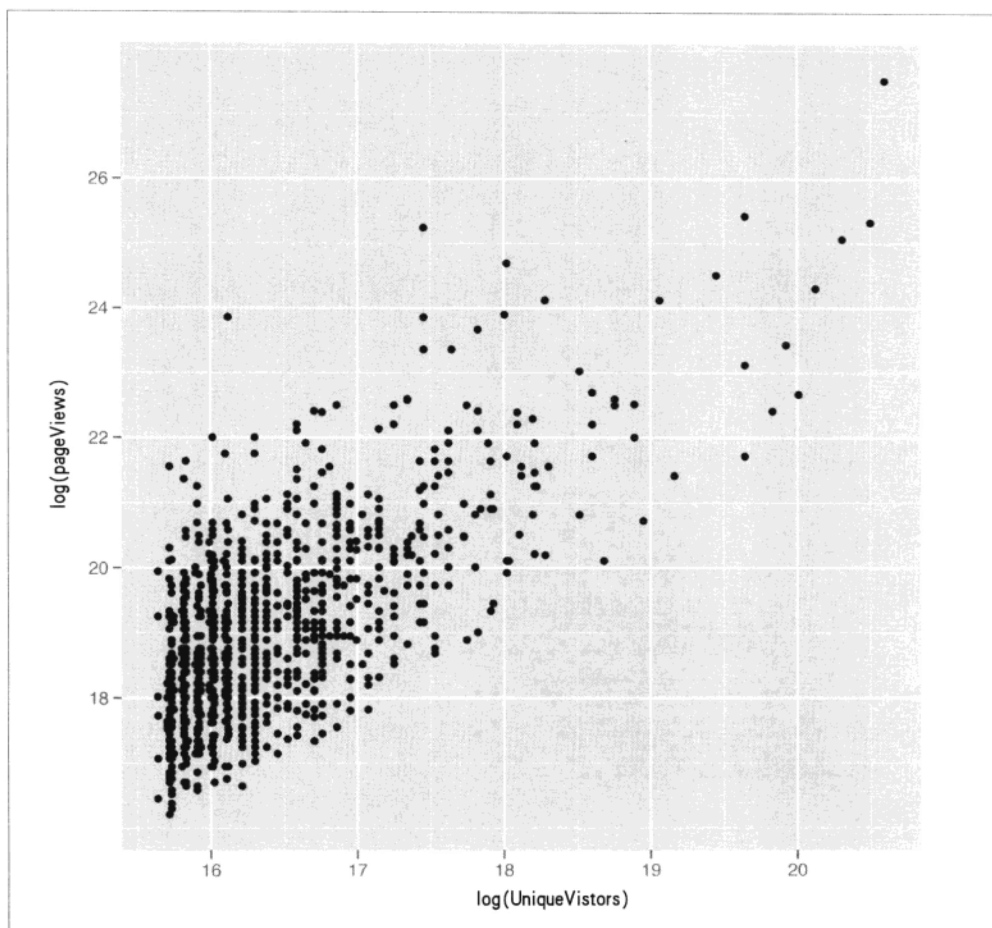


图5-9：UniqueVisitors对PageViews的log刻度散点图

画出的直线如图5-10所示，看上去很不错，那么让我们通过调用lm函数来找到定义这条直线斜率和截距的数值：

```
lm.fit <- lm(log(PageViews) ~ log(UniqueVisitors),
data = top.1000.sites)
```

现在已经拟合了这样一条直线，我们想要获得一份关于它的快速摘要信息。可以使用coef函数查看系数，或者使用residuals函数查看RMSE。不过接下来将介绍另外一个函数，这个函数产生一个更加复杂的摘要结果，我们可以一步步地解释这个结果。就把这个函数称为summary：

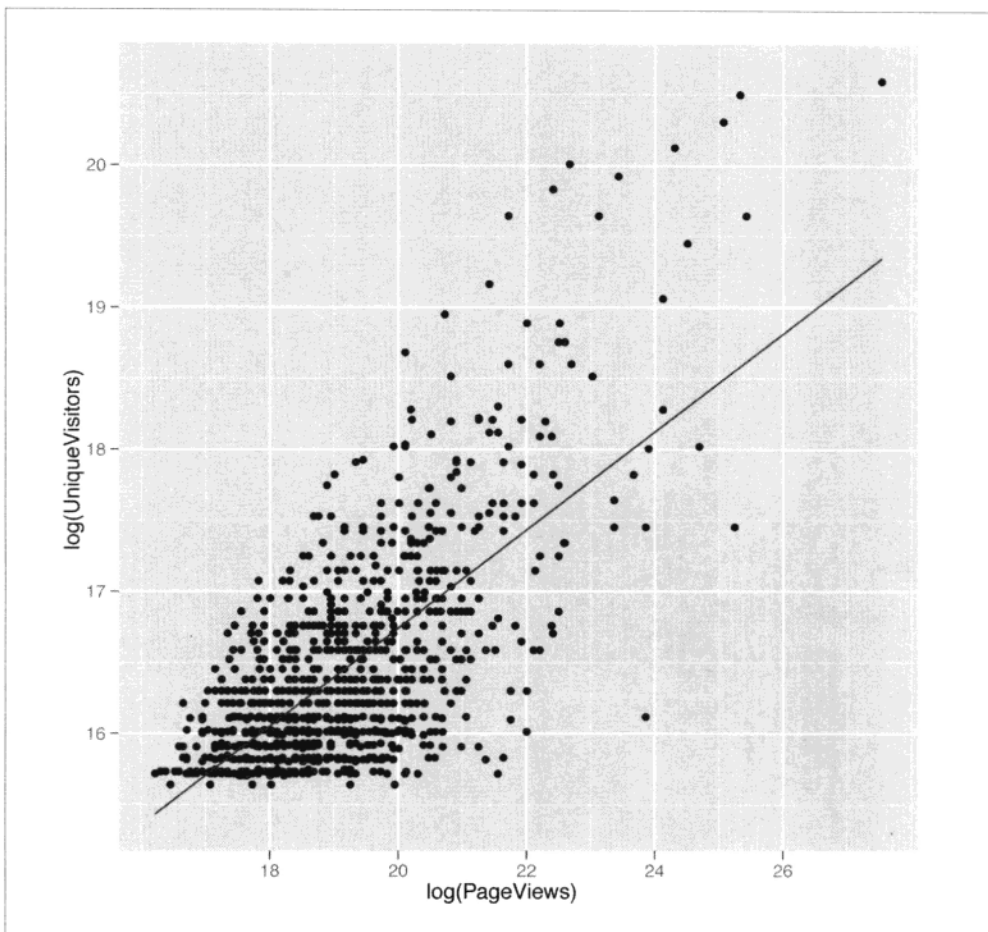


图5-10：带回归直线的UniqueVisitors对PageViews的log刻度散点图

```
summary(lm.fit)

#Call:
#lm(formula = log(PageViews) ~ log(UniqueVisitors), data = top.1000.sites)
#
#Residuals:
# Min 1Q Median 3Q Max
#-2.1825 -0.7986 -0.0741 0.6467 5.1549
#
#Coefficients:
# Estimate Std. Error t value Pr(>|t|)
#(Intercept) -2.83441 0.75201 -3.769 0.000173 ***
#log(UniqueVisitors) 1.33628 0.04568 29.251 < 2e-16 ***
#---
#Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
```

```
#Residual standard error: 1.084 on 998 degrees of freedom
#Multiple R-squared: 0.4616, Adjusted R-squared: 0.4611
#F-statistic: 855.6 on 1 and 998 DF, p-value: < 2.2e-16
```

summary函数告诉我们的第一件事情是对lm函数所做过的调用。当你使用命令行进行工作时，这不是非常有用，但是当你使用对lm进行了多次调用的大型脚本进行工作时，就变得有用了。在这种情况下，这个信息帮助你保持所有的模型都是组织良好的，以便清晰理解每个模型都有什么数据和变量输入。

Summary函数告诉我们的第二件事情是残差的分位数，如果调用quantile(residuals(lm.fit))也可以计算出来这个分位数。就个人而言，我们没发现它有多大帮助，尽管相对于观察数据的散点图，其他人也许更倾向寻找残差的最大值和最小值之间的对称性。可是，我们几乎总是可以发现图形化的表示比数字化的摘要更加容易传达信息。

接下来，summary要告诉我们比coef函数更详细的回归模型系数信息。来自coef函数的输出在结果列表中到“Estimate”这一列结束。在那之后，每一个系数都有“Std. Error”、“t-value”（t值）和“p-value”（p值）这些列。这些值用于评估我们计算的估计结果存在的不确定性，换句话说，它们就是置信度，所谓置信度就是衡量我们对计算结果能够准确描述真实世界到底有多大的信心。例如，“Std .Error”可以用于产生一个置信度为95%的系数置信区间。这个置信区间可能有些解释不清，并且偶尔还会误导他人。对置信区间所表示的范围我们可以这样表述：“在95%的情况下，构造这个区间的算法能让真实的系数值落入这个区间。”如果你对这个表述仍然不懂，那也正常：不确定性分析是非常重要的，但是比我们在本书中包含的其他内容要难很多。如果你真想深入理解这部分内容，建议你购买统计学图书，例如《All of Statistics 》[Wa03] 或者《Data Analysis Using Regression and Multilevel/Hierarchical Models》 [GH06]，并且仔细阅读。幸运的是，如果你只是想临时使用几个模型来预测某些东西，那么大可不必把注意力放在标准误差的定性差异上。

“t-value”和“p-value”（在summary函数的输出中写作“Pr(>|t|)”）两列都是用于衡量我们对真实系数不为零有多大信心的。这些用于说明我们正在考察的输入和输出之间存在真实的联系，我们对此是有信心的。例如，在此我们可以用“t-value”列来评估对PageViews和UniqueVisitors确实相关有多大把握。以作者的经验，如果你理解了它们，那这两个数字可能很有用，但是它们的用法太不好理解了，以致一些人下意识觉得永远也不可能完全弄懂统计学。如果你想知道这两个变量之间是否确实存在关联，那么应该检查估计值是否至少距离零两个标准差之外。例如，log（UniqueVisitors）的系数是1.33628，而标准差是0.04568，就是说这个系数距离零29.25306（1.33628 / 0.04568 == 29.25306）个标准差之外。如果得到的系数与零距离远在3个标准误差之上，那么你就有理由相信两个变量之间是相关的。

警告："t-value"和"p-value"用于判断两列数据之间是否真的存在关联，或者只是一种偶然现象。判断一个关系的存在是有价值的，但是理解这个关系是另一件完全不同的事情。回归模型不能帮助你理解关系。人们试图苛求用回归模型理解关系，但是，如果你想要理解两件事情相关的原因，归根结底还需要更多的信息，远非一个简单的回归模型所能提供。

确信一个输入和输出相关的传统简便方法是：为这个输入找到一个距离零为至少两个标准差之外的系数。

summary函数输出的下一部分信息是关于系数的显著性编码。那些数字旁边的星号的意思是 "t-value"有多大或者"p-value"有多小。具体来说，那些星号告诉你在"p-value"小于0.1、小于0.05、小于0.01或者小于0.001等这一系列情况下，你是否通过了前面描述的那个武断的简便判别方法。请不要担心这些让人厌烦的值，在学术界它们很常见，是从使用手工而非计算机进行统计分析的时代延续过来的。那些内容确实没有什么意思，因为你在求解估计结果距离零有多少个标准误差时并不会看到它们。你也许注意到，事实上，我们之前计算的关于log(UniqueVisitors)的"t-value"恰好是系数估计值距离零的标准差个数。"t-value"和距离零的标准差个数之间的上述关系一般来说是正确的，因此建议你根本不需要使用"p-value"。

我们得到的最后一部分信息是关于从数据中拟合得到的线性模型的预测能力。第一个是"Residual Standard Error"，很简单，就是我们使用sqrt(mean(residuals(lm.fit) ^ 2))计算出来的RMSE。"degree of freedom"（自由度）是指下面的概念，我们在分析中使用的数据点至少要有两个，才能有效地拟合出两个系数，这两个系数是截距和log*UnqueVisitors*的系数。998[译注1]这个数字也跟自由度是相关的，因为如果你用500个系数去拟合1000个数据点，那么就算RMSE值较低也不值得称赞。当数据较少而使用的系数却较多时，就是过拟合的一种形式，这将在第6章中深入讨论。

然后，我们看到的是"Multiple R-squared"。这是标准的"R平方"，这在前面已描述过，它指明我们的数据中存在的变化有多少已经被模型所解释。在这里我们使用模型解释了46%的变化，这已经相当好了。"Adjusted R-squared"是第二个度量，它根据你使用的系数的个数调整"Multiple R-squared"，系数的个数越多，"Multiple R-squared"就会得到越大的惩罚。在实践中，我个人倾向于忽略这个值，因为我认为它有点像个特设参数，但是有很多人非常喜欢它。

最后，你将看到的最后一个信息是"F-statistic"。这是关于你的模型相对于仅仅使用均值来做预测所获得的效果提升的一个度量。它是"R平方"的一个替代方案，可以用来

译注1：数据点个数与系数个数的差值。

计算“p-value”。因为我们认为“p-value”通常是有欺骗性的，所以我们建议不要太过相信“F-statistic”。如果你完全理解用于计算“p-value”的原理，那么“p-value”是有其用途的，但是它们会提供一种虚假的安全感，这种感觉将使你忘记模型效果的黄金标准是：它在未知数据[译注2]上的预测能力，而不是你的模型在用于拟合它的数据上的效果。我们将在第6章讨论如何评估你的模型预测新数据的能力。

讨论summary函数的输出结果已经让我们离题万里，不过我们不想仅仅用UniqueVisitors来关联PageViews，还想引入一些其他类型的信息。我们也将包含HasAdvertising 和IsEnglish来看看当给模型更多的输入时，会发生什么：

```
lm.fit <- lm(log(PageViews) ~ HasAdvertising + log(UniqueVisitors) + InEnglish,
            data = top.1000.sites)
summary(lm.fit)

#Call:
#lm(formula = log(PageViews) ~ HasAdvertising + log(UniqueVisitors) +
# InEnglish, data = top.1000.sites)
#
#Residuals:
# Min 1Q Median 3Q Max
#-2.4283 -0.7685 -0.0632 0.6298 5.4133
#
#Coefficients:
# Estimate Std. Error t value Pr(>|t|)
#(Intercept) -1.94502 1.14777 -1.695 0.09046 .
#HasAdvertisingYes 0.30595 0.09170 3.336 0.00088 ***
#log(UniqueVisitors) 1.26507 0.07053 17.936 < 2e-16 ***
#InEnglishNo 0.83468 0.20860 4.001 6.77e-05 ***
#InEnglishYes -0.16913 0.20424 -0.828 0.40780
#---
#Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
#
#Residual standard error: 1.067 on 995 degrees of freedom
#Multiple R-squared: 0.4798, Adjusted R-squared: 0.4777
#F-statistic: 229.4 on 4 and 995 DF, p-value: < 2.2e-16
```

我们再一次看到summary函数输出了曾对lm函数做过的调用，并且输出了残差。这次输出结果中新增加的是在这个更复杂的回归模型里包含的所有变量的系数。这里又看到了截距，在输出结果中是（Intercept）。下一项和我们之前看到的都不一样，这是因为我们的模型现在包含了一个因子。当在回归模型中使用一个因子时，模型必须做出决定把因子的一种取值情况作为截距的一部分，而因子的其他取值情况在模型中显式地呈现出来。在这里可以看到因子HasAdvertising（是否有广告）被用于建模，因此，对于那些HasAdvertising == 'Yes'（有广告）的网站来说，这个因子取值从截距中分离出，而对于那些HasAdvertising == 'No'（无广告）的网站来说，这个因子取值包含进截距中。

译注2：这里的未知数据是指没有用来拟合的数据。

换种说法来讲，截距就是对一个没有广告并且log（UniqueVisitors）值为零的网站的流量预测，这种情况发生在UniqueVisitors取值为1时。

可以看到对于InEnglish也发生了同样的逻辑，这个因子有很多NA值，因此对于这个虚拟变量来说实际上有三个级别的取值：NA、No和Yes。在这种情况下，R语言默认把NA值包含进回归模型的截距中，并且分别对No和Yes两个取值拟合系数。

现在我们已经考虑了如何将这些因子作为回归模型的输入，让我们单独比较之前用过的三个输入，来看看当只使用其中一个时，哪个输入具有最强的预测能力。要做到这一点，我们可以单独提取每个summary函数的R平方值：

```
lm.fit <- lm(log(PageViews) ~ HasAdvertising,
             data = top.1000.sites)
summary(lm.fit)$r.squared
#[1] 0.01073766

lm.fit <- lm(log(PageViews) ~ log(UniqueVisitors),
             data = top.1000.sites)
summary(lm.fit)$r.squared
#[1] 0.4615985
lm.fit <- lm(log(PageViews) ~ InEnglish,
             data = top.1000.sites)
summary(lm.fit)$r.squared
#[1] 0.03122206
```

正如你所看到的，HasAdvertising只解释了1%的方差，UniqueVisitors解释了46%，而InEnglish解释了3%。在实践中，当输入容易获得时，值得将所有输入都包含进一个预测模型，但是如果HasAdvertising是难以通过程序获得的，那么建议把它从一个拥有其他更具预测能力的输入模型中去掉。

定义相关性

到目前为止已经介绍了线性回归模型，现在简单谈谈另一个主题——相关性。从严格意义上说，如果两个变量之间可以用一条直线描述，那么它们就存在相关性。换句话说，相关性用来衡量线性回归模型对两个变量之间关系建模的好坏。值为0的相关性表明没有一条我们感兴趣的直线能将两个变量联系起来。值为1的相关性表明有一条完美的正向直线（上升）把两个变量联系起来。值为-1的相关性表明有一条完美的负向直线（下降）把两个变量联系起来。

为了阐明这些概念，接下来看一个简短的例子。首先，我们将产生一些不是严格线性的数据并且画出它们。

```
x <- 1:10
y <- x ^ 2
```

```
ggplot(data.frame(X = x, Y = y), aes(x = X, y = Y)) +
geom_point() +
geom_smooth(method = 'lm', se = FALSE)
```

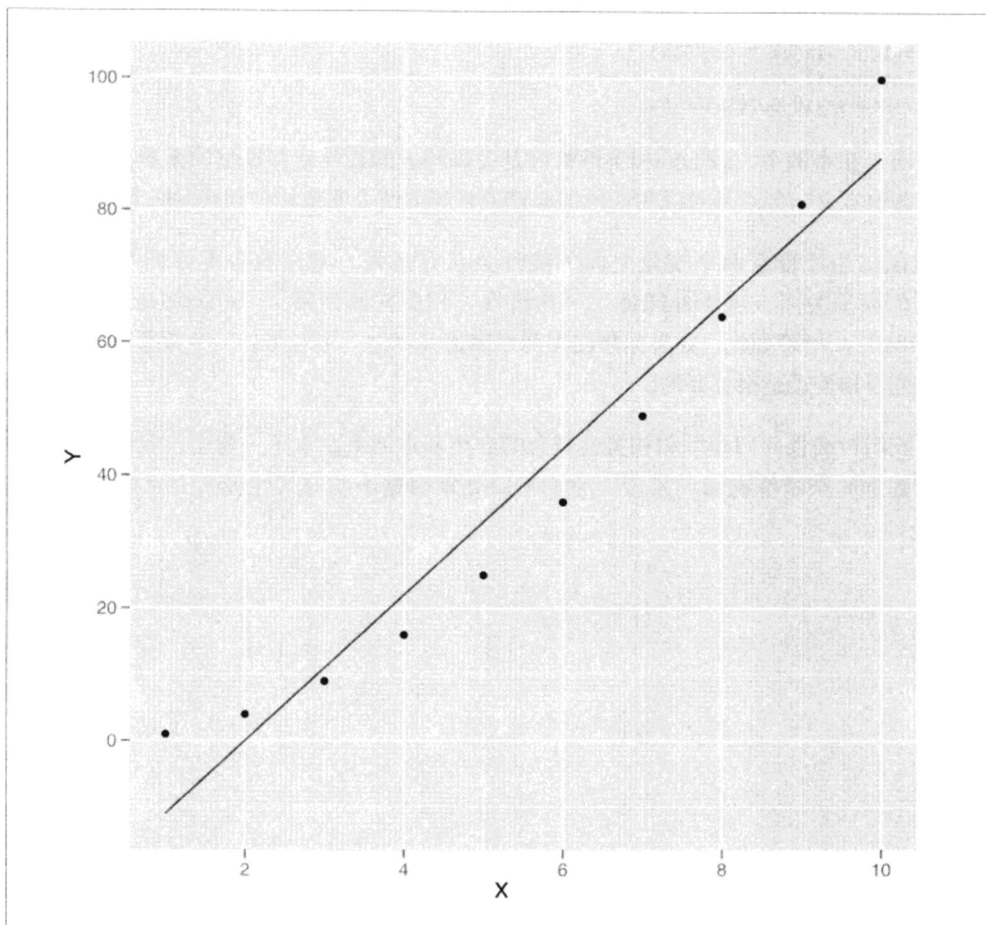


图5-11：直线展示了X和Y之间的非完美线性关系

样本数据如图5-11所示。如你所见，使用`geom_smooth`函数画出的直线并没有通过所有的点，因此x和y之间的关系不可能是完美的线性。但是有多么接近完美呢？为了回答这个问题，我们使用`cor`函数来比较相关性：

```
cor(x, y)
#[1] 0.9745586
```

在这里我们可以看到，x和y可以由一条相当好的直线关联起来，`cor`函数可以用来估计这个关系有多强。在这个例子中，这个估计值是0.97，几乎是1。

我们怎样才能自己计算相关性而不是用cor函数呢？我们可以用lm函数，但是先得对x和y进行刻度变换。调整刻度，先是减去两个变量的均值，再除以标准差。在R语言中，你可以使用scale函数执行刻度变换：

```
coef(lm(scale(y) ~ scale(x)))
# (Intercept) scale(x)
#-1.386469e-16 9.745586e-01
```

可以看出，在本例中，x和y的相关性恰好是关联两者的线性回归模型的系数。计算相关性具有普遍意义，你总是可以用线性回归模型来理清两个变量相关到底意味着什么。

相关性仅仅用于度量两个变量之间的线性关系有多强，它没有告诉我们两个变量之间有任何因果关系。这恰好印证了一句格言“相关并非因果”（correlation is not causation）。尽管如此，如果你想使用两件事情其中之一去对另外一个做预测，那么知道它们是否相关还是很重要的。

至此，我们对线性回归模型和相关性概念的介绍基本结束。在下一章中，我们将展示如何使用更加复杂精练的回归模型，它们可以处理数据中的非线性模式并且同时预防过拟合。

第6章

正则化：文本回归

数据列之间的非线性关系：超越直线

我们在第5章里说过，线性回归假设两个变量之间的关系可以用直线来表示，事实上，如果两个变量之间的关系并不适合用直线来表示，而我们非要用线性回归来拟合它们也未尝不可。不过，让我们来看看这会导致什么问题，假设你有图6-1a的数据。

显然，从这张散点图来看，X和Y之间的关系用直线来描述并不合适。事实上，若把拟合出来的回归直线画出来，就可以知道用直线来描述X和Y之间的关系会出什么问题，详见图6-1b。

如果用直线来拟合X和Y的关系，我们对y的预测值就存在系统误差：当X较小或较大时，我们高估了Y的值；而其余的时候，我们又低估了Y的值。这可以从图6-1c中清楚地看出。在图6-1c中，可以看到原数据集合的所有结构，而这种结构完全不能被线性回归模型所描述。

为了对数据进行一个光滑的非线性拟合，可以使用更复杂的Generalized Additive Model（GAM，广义加性模型）统计模型，通过使用ggplot2程序包中的gemo_smooth函数，并使用默认的method参数，就可以拟合GAM模型：

```
set.seed(1)

x <- seq(-10, 10, by = 0.01)
y <- 1 - x^2 + rnorm(length(x), 0, 5)
ggplot(data.frame(X = x, Y = y), aes(x = X, y = Y)) +
geom_point() +
 geom_smooth(se = FALSE)
```

结果如图6-1d所示，我们要拟合的数据集其实是一条曲线而不是直线。

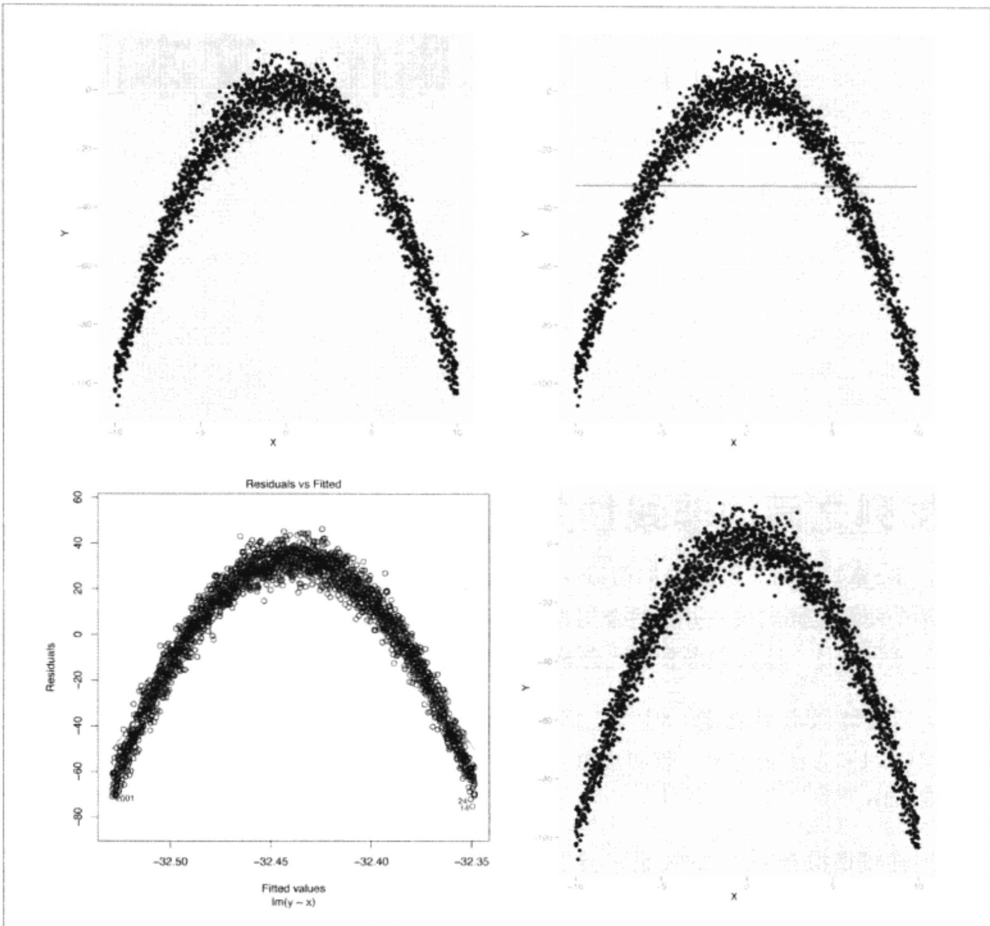


图6-1：为非线性数据建模：a）图形化非线性关系；b）非线性关系与线性拟合；c）结构化的残差；d）广义加性模型的结果

那么，我们可以将数据拟合成曲线吗？这正是线性回归的微妙之处：尽管线性回归只能将输入拟合成直线，但是你可以使用非线性函数将输入转化成新的输入。例如，你可以使用下面的R语言代码来将原始输入x转化成原始输入的平方：

```
x.squared <- x ^ 2
```

然后，你可以将数据中的Y与x.squared进行拟合，你会看到一个与之前非常不同的拟合形状：

```
ggplot(data.frame(XSquared = x.squared, Y = y), aes(x = XSquared, y = Y)) +
```

```
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE)
```

正如图6-2所示，将y与x.squared进行拟合之后的图形非常接近一条直线。从本质上说，我们已经将原先的非线性问题转化成一个新的线性回归问题了。类似这种通过对输入进行转换，将一个复杂的非线性问题转换成一个线性问题的方法在机器学习问题中非常常见。事实上，这也是将在第12章中提到的kernel trick（核方法）的本质思想。我们比较一下使用x和x.squared进行线性回归的R^2值，就能明白这个简单的转换在准确率方面提升了多少。

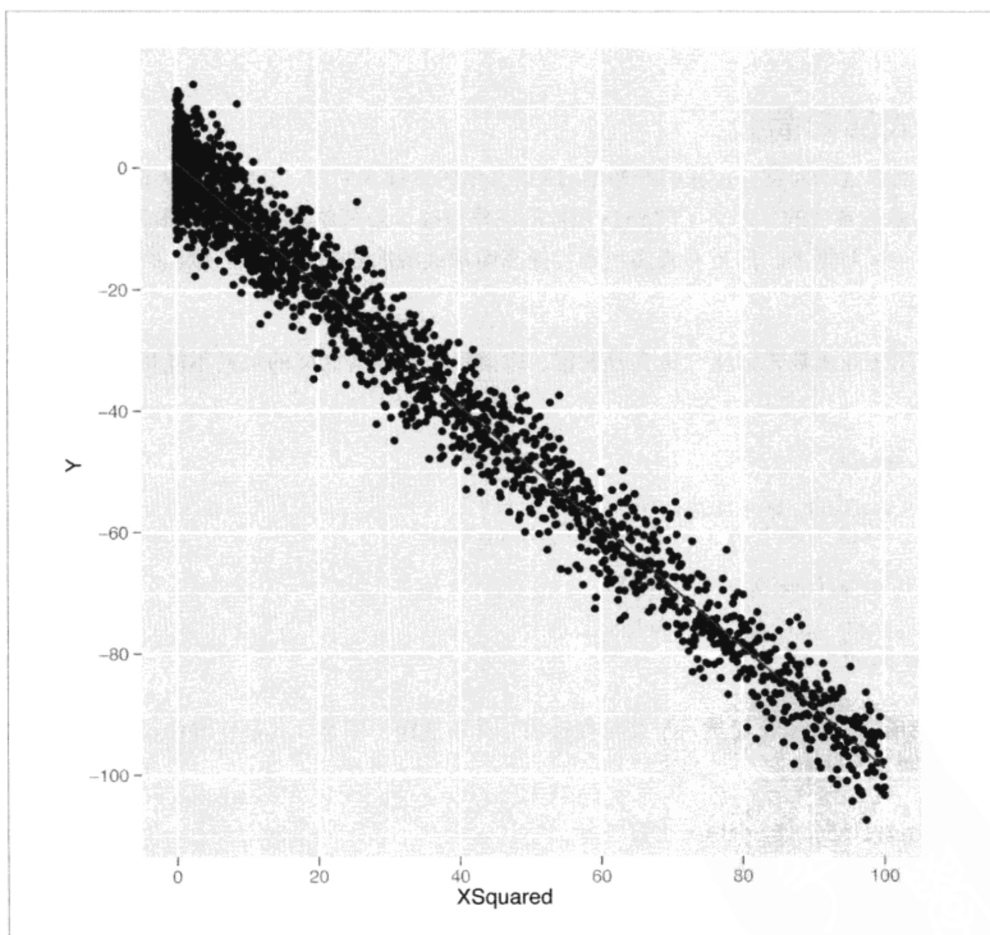


图6-2：非线性回归

```
summary(lm(y ~ x))$r.squared
#[1] 2.973e-06
```

```
summary(lm(y ~ x.squared))$r.squared
#[1] 0.9707
```

我们所能解释的数据占比从几乎0%提升到97%。对于一个如此简单的改进来说，这个提升非常巨大。通常，我们会好奇，如果用比直线更复杂的模型来拟合数据，准确率能提高多少。因为相关的数学原理非常复杂，所以这个问题的解答已超出了本书的讨论范围，其实，只要选择的曲线足够复杂，你就可以拟合两个变量之间可能存在的任何一种关系。其中一种拟合复杂形状的方法我们将会在本章的下一节中介绍——多项式回归（polynomial regression）。但是，多项式拟合带来的灵活性并不是没有代价的，它会让我们拟合数据中的噪声，而不是真正想要拟合的数据模型。因此，如果打算用多项式拟合来替代线性拟合，必须遵守一些额外的准则，本章的余下内容将会关注这些准则。

多项式回归简介

带着前面提过的问题，让我们开始学习R语言中的多项式回归，在R语言中多项式回归是由`poly`函数实现的。想要了解`poly`函数的工作原理，最简单的方式就是将它应用在一份简单的实例数据上，然后看看假如我们提高多项式的次数来“更好”地拟合数据时会发生什么。

我们会用正弦函数来创建一份实验数据，以确保变量x和y之间的关系不能用简单的直线来描述。

```
set.seed(1)

x <- seq(0, 1, by = 0.01)
y <- sin(2 * pi * x) + rnorm(length(x), 0, 0.1)

df <- data.frame(X = x, Y = y)

ggplot(df, aes(x = X, y = Y)) +
geom_point()
```

如图6-3所示的数据，显然一个简单线性拟合并不适用。不过，还是让我们试一下线性模型，看看会发生什么：

```
summary(lm(Y ~ X, data = df))

#Call:
#lm(formula = Y ~ X, data = df)
#
#Residuals:
# Min 1Q Median 3Q Max
#-1.00376 -0.41253 -0.00409 0.40664 0.85874
#
#Coefficients:
```

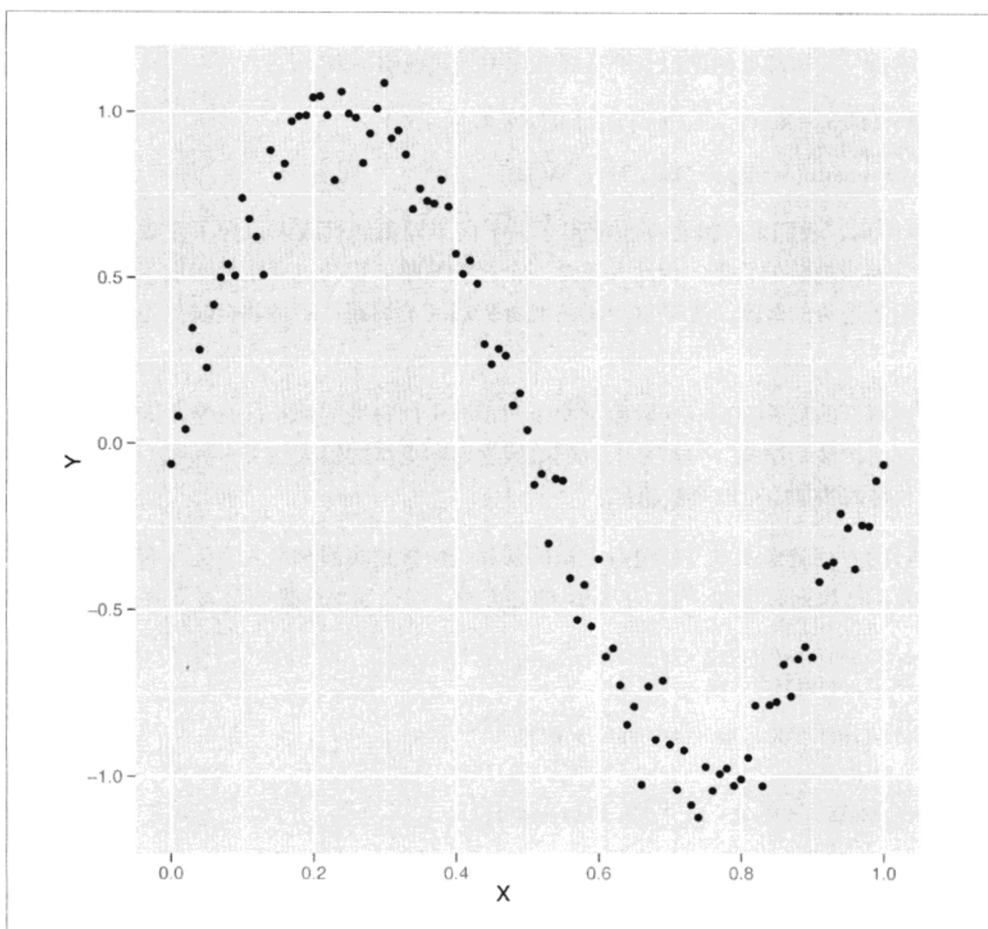


图6-3：非线性数据

```
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept) 0.94111   0.09057   10.39   <2e-16 ***
#X           -1.86189   0.15648   -11.90  <2e-16 ***
#---

#Signif. codes: 0  '***'  0.001  '**'  0.01  '*'  0.05  '.'  0.1  ' '  1
#
#Residual standard error: 0.4585 on 99 degrees of freedom Multiple R-squared: 0.5885,
#Adjusted R-squared: 0.5843 F-statistic: 141.6 on 1 and 99 DF, p-value: < 2.2e-16
```

结果出乎意料，针对实验数据集使用线性回归，线性模型可以解释其中60%的数据，尽管我们清楚，对正弦波数据而言，线性回归并不是一个好模型。我们同样知道，一个好的模型应该至少能解释90%的数据，但我们仍然要指出线性模型究竟是怎么做到能解释

其中60%的数据的。要回答这个问题，最好使用geom_smooth函数，并且指定它的method参数为‘lm’，这样就能把拟合的直线在图中显式地画出来：

```
ggplot(data.frame(X = x, Y = y), aes(x = X, y = Y)) +
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE)
```

如图6-4所示，我们发现线性模型使用了一个向下倾斜的直线来拟合了正弦波数据中间那部分呈递减趋势的数据。但这并不是一个好的策略，因为这系统地无视了数据头尾两部分呈递增趋势的数据。如果这个正弦波再扩展一个周期，R^2值将会突然下降到接近于0%。

我们发现默认的线性回归在实验数据集上只拟合中间递减的那小部分特定的数据，并没能发现真实的波形结构。但是如果我们给线性回归更多的输入会怎么样呢？它能发现它学习的数据其实是一个波形数据吗？

一个可行的方法就像本章开始时我们做的那样，给数据集增加输入。为了有更多的回旋余地，我们这次同时增加x的平方项和x的立方项。你会发现这显著提高了预测准确率：

```
df <- transform(df, X2 = X ^ 2)
df <- transform(df, X3 = X ^ 3)

summary(lm(Y ~ X + X2 + X3, data = df))

#Call:
#lm(formula = Y ~ X + X2 + X3, data = df)
#
#Residuals:
# Min 1Q Median 3Q Max
#-0.32331 -0.08538 0.00652 0.08320 0.20239
#
#Coefficients:
# Estimate Std. Error t value Pr(>|t|)
#(Intercept) -0.16341 0.04425 -3.693 0.000367 ***

#X 11.67844 0.38513 30.323 < 2e-16 ***
#X2 -33.94179 0.89748 -37.819 < 2e-16 ***
#X3 22.59349 0.58979 38.308 < 2e-16 ***
#---
#Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#
#Residual standard error: 0.1153 on 97 degrees of freedom
#Multiple R-squared: 0.9745, Adjusted R-squared: 0.9737
#F-statistic: 1235 on 3 and 97 DF, p-value: < 2.2e-16
```

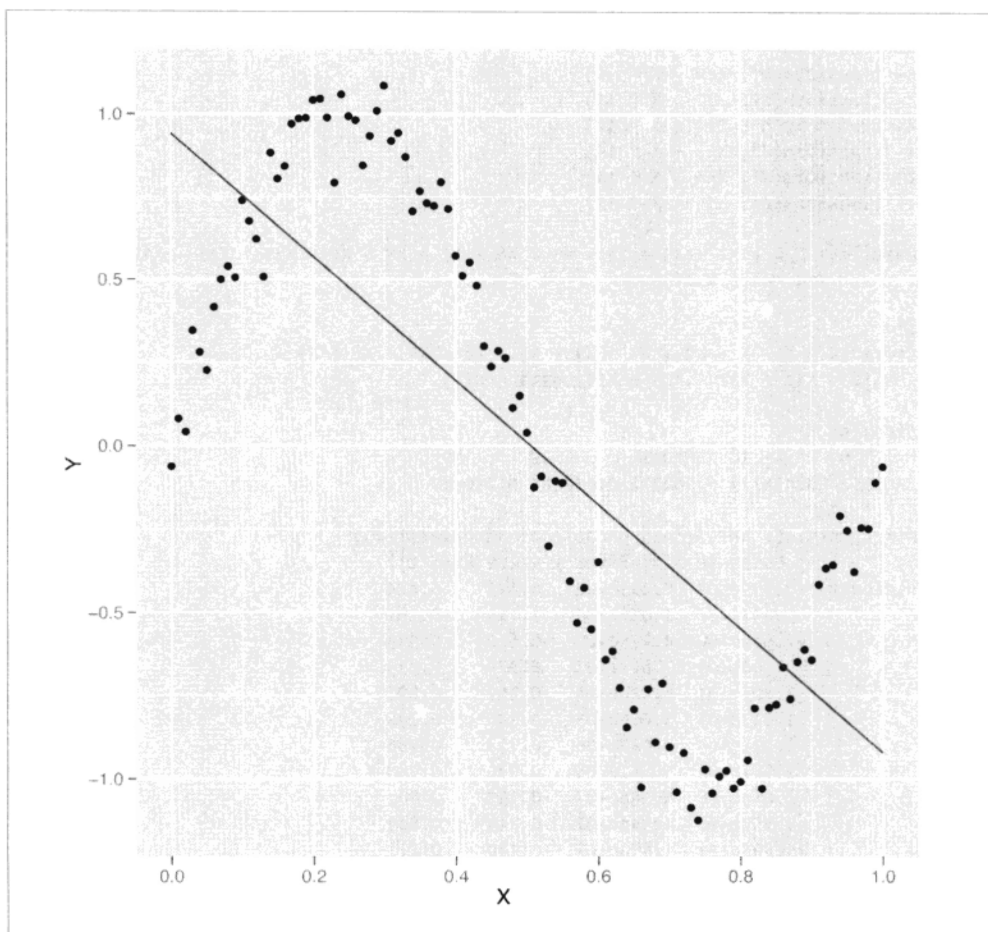


图6-4：非线性数据的光滑线性拟合

只增加了两个输入特征，R^2值就从60%提升到97%，这的确是巨大的提升。而且，从原则上说，只要我们愿意，可以继续增加更多的x高次方项到数据集里来。但是，随着更多的x高次方项的加入，我们最终会发现，输入特征的个数已经超过了样本的个数——这通常是一个比较麻烦的事情，因为这意味着，原则上，我们可以完美地拟合训练数据。但到那时我们会遇到一个比较棘手的问题：新增的高次方列数据与之前的列数据之间太相关，以至于`lm`函数不能正常拟合了。接下来的`summary`函数输出中我们会看到一个叫做奇异点（singularity）的问题。

```
df <- transform(df, X4 = X ^ 4)
df <- transform(df, X5 = X ^ 5)
df <- transform(df, X6 = X ^ 6)
df <- transform(df, X7 = X ^ 7)
```

```
df <- transform(df, X8 = X ^ 8)
df <- transform(df, X9 = X ^ 9)
df <- transform(df, X10 = X ^ 10)
df <- transform(df, X11 = X ^ 11)
df <- transform(df, X12 = X ^ 12)
df <- transform(df, X13 = X ^ 13)
df <- transform(df, X14 = X ^ 14)
df <- transform(df, X15 = X ^ 15)

summary(lm(Y ~ X + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 + X12 + X13 +
           X14, data = df))

#Call:
#lm(formula = Y ~ X + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 +
#     X10 + X11 + X12 + X13 + X14, data = df)
#
#Residuals:
#      Min        1Q    Median       3Q      Max
#-0.242662 -0.038179 0.002771 0.052484 0.210917
#
#Coefficients: (1 not defined because of singularities)
#              Estimate Std. Error t value Pr(>|t|)
#(Intercept) -6.909e-02  8.413e-02  -0.821    0.414
#X            1.494e+01  1.056e+01   1.415    0.161
#X2          -2.609e+02  4.275e+02  -0.610    0.543
#X3           3.764e+03  7.863e+03   0.479    0.633
#X4          -3.203e+04  8.020e+04  -0.399    0.691
#X5           1.717e+05  5.050e+05   0.340    0.735
#X6          -6.225e+05  2.089e+06  -0.298    0.766
#X7           1.587e+06  5.881e+06   0.270    0.788
#X8          -2.889e+06  1.146e+07  -0.252    0.801
#X9           3.752e+06  1.544e+07   0.243    0.809
#X10         -3.398e+06  1.414e+07  -0.240    0.811
#X11          2.039e+06  8.384e+06   0.243    0.808
#X12         -7.276e+05  2.906e+06  -0.250    0.803
#X13          1.166e+05  4.467e+05   0.261    0.795
#X14                 NA         NA      NA       NA
#
#Residual standard error: 0.09079 on 87 degrees of freedom
#Multiple R-squared: 0.9858, Adjusted R-squared: 0.9837
#F-statistic: 465.2 on 13 and 87 DF, p-value: < 2.2e-16
```

造成奇异点问题的原因是：新增x的高次方特征与之前x低次方特征之间太相关，以致线性回归算法不能正常执行——无法为每一个特征找到合适的权重系数了。幸运的是，我们可以从数学文献中找到解决这个问题的方法：并不只是简单地增加x的高次方项，而是增加更复杂的关于x的高次方多项式，虽然它们的作用类似x的纯高次方项，但是并不像x和x^2那样相关。这些更加复杂的x的高阶多项式称作正交多项式[注1]（orthogonal polynomials），在R语言里面，你可以使用poly函数生成正交多项式。不再是简单地将x的1～14次方项直接加到数据集中，而是使用函数poly(X, degree = 14)，这个函数返回

注1： 正交意味着不相关。

的结果类似于X + X^2 + X^3 + ... + X^14，但是它们互相正交，因此也就不会在调用lm函数时产生奇异点问题了。

为了确信poly运行正常，可以通过poly函数的返回值为参数运行lm函数，然后你就会看到函数确实返回了x的1～14次方项的对应权重。

```
summary(lm(Y ~ poly(X, degree = 14), data = df))

#Call:
#lm(formula = Y ~ poly(X, degree = 14), data = df)
#
#Residuals:
#      Min        1Q    Median        3Q       Max
#-0.232557 -0.042933 0.002159 0.051021 0.209959
#
#Coefficients:
#                       Estimate Std. Error t value Pr(>|t|)
#(Intercept)            0.010167   0.009038   1.125   0.2638
#poly(X, degree = 14)1 -5.455362   0.090827 -60.063  < 2e-16 ***
#poly(X, degree = 14)2 -0.039389   0.090827  -0.434   0.6656
#poly(X, degree = 14)3 4.418054    0.090827  48.642  < 2e-16 ***
#poly(X, degree = 14)4 -0.047966   0.090827  -0.528   0.5988
#poly(X, degree = 14)5 -0.706451   0.090827  -7.778 1.48e-11 ***
#poly(X, degree = 14)6 -0.204221   0.090827  -2.248   0.0271 *
#poly(X, degree = 14)7 -0.051341   0.090827  -0.565   0.5734
#poly(X, degree = 14)8 -0.031001   0.090827  -0.341   0.7337
#poly(X, degree = 14)9  0.077232   0.090827   0.850   0.3975
#poly(X, degree = 14)10 0.048088   0.090827   0.529   0.5979
#poly(X, degree = 14)11 0.129990   0.090827   1.431   0.1560
#poly(X, degree = 14)12 0.024726   0.090827   0.272   0.7861
#poly(X, degree = 14)13 0.023706   0.090827   0.261   0.7947
#poly(X, degree = 14)14 0.087906   0.090827   0.968   0.3358
#---
#Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
#Residual standard error: 0.09083 on 86 degrees of freedom
#Multiple R-squared: 0.986, Adjusted R-squared: 0.9837
#F-statistic: 431.7 on 14 and 86 DF, p-value: < 2.2e-16
```

通常来说，poly函数提供了很强大的拟合能力，而且数学上已经证明，多项式回归可以帮你拟合各种各样形状复杂的数据。

但它未必是件好事。如果你一边增加degree参数，一边注意观测得到的模型的形状，就会发现，通过poly函数增加的次数可能会带来麻烦。在接下来的例子中，我们分别用1次、3次、5次和25次多项式来生成模型，结果见图6-5。

```
poly.fit <- lm(Y ~ poly(X, degree = 1), data = df)
df <- transform(df, PredictedY = predict(poly.fit))

ggplot(df, aes(x = X, y = PredictedY)) +
  geom_point() +
```

```
  geom_line()

poly.fit <- lm(Y ~ poly(X, degree = 3), data = df)
df <- transform(df, PredictedY = predict(poly.fit))

ggplot(df, aes(x = X, y = PredictedY)) +
  geom_point() +
  geom_line()

poly.fit <- lm(Y ~ poly(X, degree = 5), data = df)
df <- transform(df, PredictedY = predict(poly.fit))

ggplot(df, aes(x = X, y = PredictedY)) +
  geom_point() +
  geom_line()

poly.fit <- lm(Y ~ poly(X, degree = 25), data = df)
df <- transform(df, PredictedY = predict(poly.fit))

ggplot(df, aes(x = X, y = PredictedY)) +
  geom_point() +
  geom_line()
```

我们可以无限制地将次数增加上去，但是看一下得到的模型的形状我们就会发现，最终拟合出来的模型形状不再像一个正弦波了，而变得歪歪扭扭。造成这个问题的原因就是我们使用了数据所无法支持的太过于复杂的模型。当我们的模型的次数维持在1、3或者5的时候，情况还好，但是当次数增加到25的时候，情况就不妙了。我们目前遇到的这个问题的本质就是过拟合（overfitting）。数据越多，就越能够使用强大而复杂的模型。但是对于一定的数据量来说，总归有某些模型实在是太过于复杂了。我们该怎么避免过拟合呢？我们又该怎么知道当前的模型已经过拟合了，并赶紧悬崖勒马呢？答案是在机器学习领域中最重要的两个技术手段——交叉验证（cross-validation）与正则化（regularization）。

避免过拟合的方法

在谈及如何避免"过拟合"之前，我们需要先对过拟合有一个严格的定义。我们认为，过拟合是指一个模型拟合了部分噪声，而不是真正数据。但是，如果不能区分什么是噪声什么是真正数据，我们又如何判定是否发生了过拟合呢？

这里的诀窍在于如何定义"真实数据"。对我们而言，一个模型的好坏取决于它能否准确预测未来的未知数据。由于没有时光机，因此我们没有"未来"的数据，只有过去的数据。幸运的是，我们有变通的方法：可以把过去的数据分成两份，用其中一份拟合模型，用另一份数据模拟"将来的"数据。

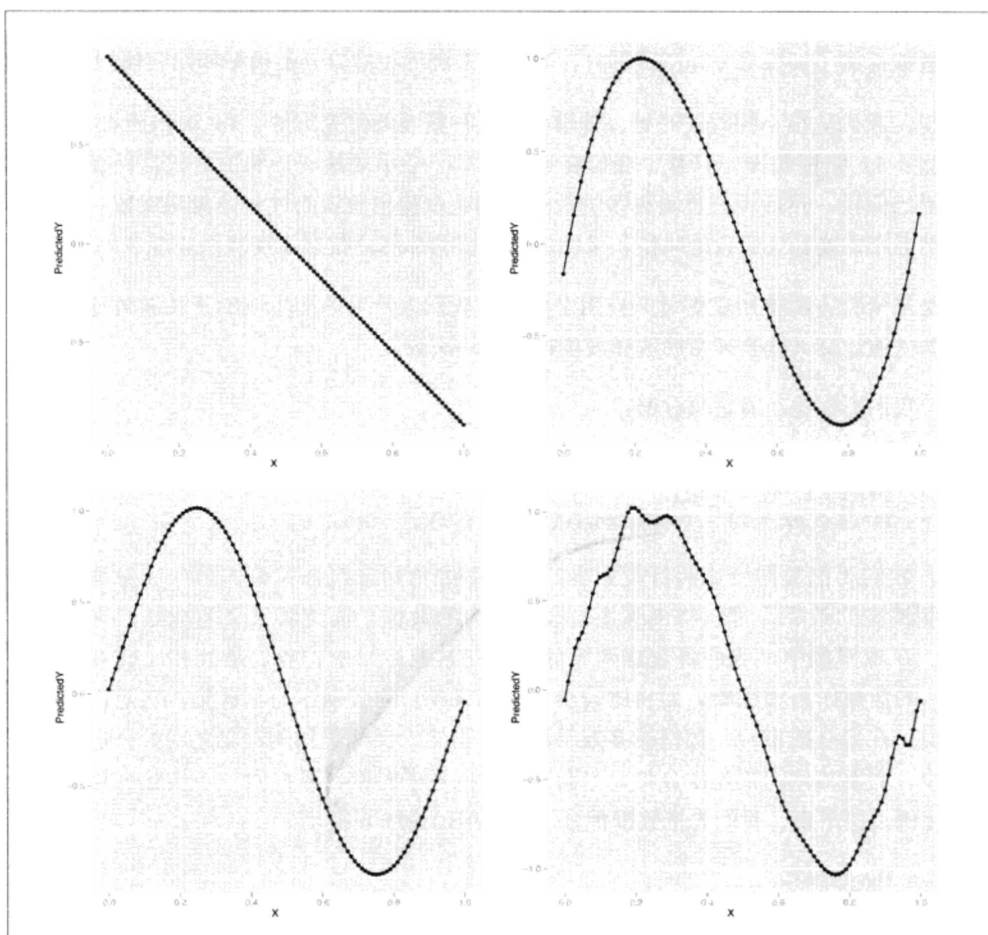


图6-5：多项式回归：a）1次多项式；b）3次多项式；c）5次多项式；d）25次多项式

让我们举一个简单的例子：假设想建立一个预测气温的模型。我们想根据1～6月的数据来预测7月的数据。如果想知道哪一个模型最好，我们可以通过1～5月的数据来训练模型，然后将模型应用在6月上来看预测的准确率如何。现在，如果我们假定上述的模型拟合过程发生在5月底，那对于当时（5月底）的我们而言，就真的是在拿模型预测“未来”（即6月）的数据了。通过这个例子，我们就知道用类似这种数据拆分的方法，可以用“未来”的数据来测试模型。

交叉验证的核心思想，就是在模型拟合的过程中并不使用全部的历史数据，而是保留一部分数据，用来模拟未来的数据，对模型进行检验。

如果你曾经通读过第3章和第4章，就不会对这种做法感到陌生。在那些案例中，我们分别把数据拆成了训练部分和测试部分，分别用来训练和检验分类模型和回归模型。

可以说，我们从很小的时候开始，就学会了这种科学的研究方法：1）提出假设；2）收集数据；3）验证假设。只是，我们在这里变了一个小戏法——并不是根据已有的全部数据提出假设，然后再去收集更多的数据。而是在提出假设的过程中故意保留一部分数据，然后在需要验证假设的时候，又把之前保留的那部分数据像变戏法般地拿出来。

在将交叉检验方法使用在更复杂应用之前，让我们先针对前面生成的正弦波数据来演示一下如何通过交叉检验来帮助选择多项式回归的次数。

首先，我们再创建一遍正弦数据：

```
set.seed(1)
x <- seq(0, 1, by = 0.01)
y <- sin(2 * pi * x) + rnorm(length(x), 0, 0.1)
```

然后，我们要把数据分成两份数据集：一份训练集用来拟合模型，另外一份测试集用来检测模型的效果。可以将训练集当成是过去的数据，而将测试集当成是“未来”的数据。在本案例中，我们将数据两等分。而在其他应用中，我们最好将比较多（比如80%）的数据作为训练集，而将比较少（比如20%）的数据作为测试集，这是因为，通常来说，在拟合模型时，数据越多效果越好。当然，训练集和测试集之间最合适的比例是多少，需要具体问题具体分析，当面对实际问题的时候，最好做一下实验来决定最合适的比例。接下来，我们先把数据拆分，然后再讨论细节：

```
n <- length(x)

indices <- sort(sample(1:n, round(0.5 * n)))
training.x <- x[indices]
training.y <- y[indices]

test.x <- x[-indices]
test.y <- y[-indices]

training.df <- data.frame(X = training.x, Y = training.y)
test.df <- data.frame(X = test.x, Y = test.y)
```

首先创建一个随机向量，并拿它当做数组下标用来从原始数据中选取训练集。随机拆分训练集和测试集是一个好主意，你肯定不希望训练集和测试集有系统性地差异——比如，你可能拿较小的x来训练，而拿较大的x来测试。R语言的`sample`函数可以提供随机性，它可以从一个给定的向量中进行随机采样。在上例中，我们创建了一个从1到n的整数向量，然后从中随机采样一半。一旦确定了下标的值，就可以使用R语言的向量索引运算符把训练集和测试集分开存放。最后，为了更方便地使用`lm`函数，可以创建一个数据框来存放数据。

一旦将数据拆分为训练集和测试集之后，可以尝试不同的拟合次数，来看看哪种多项式回归效果最好。通过使用RMSE来度量效果。为了增加代码的可读性，我们专门写一个rmse函数：

```
rmse <- function(y, h)
{
  return(sqrt(mean((y - h) ^ 2)))
}
```

接下来，我们从1~12对不同的次数进行循环：

```
performance <- data.frame()

for (d in 1:12)
{
   poly.fit <- lm(Y ~ poly(X, degree = d), data = training.df)

   performance <- rbind(performance,
                        data.frame(Degree = d,
                                   Data = 'Training',
                                   RMSE = rmse(training.y, predict(poly.fit))))

  performance <- rbind(performance,
                       data.frame(Degree = d,
                                  Data = 'Test',
                                  RMSE = rmse(test.y, predict(poly.fit,
                                              newdata = test.df))))
}
```

在这个循环的每次迭代中，我们将测试数据training.df拟合到一个d次多项式，然后分别将模型应用到training.df和test.df上得到对应的训练误差和测试误差。我们将结果存放在一个数据框当中，这样当循环结束的时候，可以非常方便地分析结果。

警告： 在这里，我们使用了一个与过去不太一样的创建数据框的方法。首先将performance变量初始化成一个空的数据框，然后通过rbind函数循环地把行数据加入数据框。正如前面提过的，在R语言里面，我们可以用各种各样不同的方法来实现同一个数据操作目的，不过，通常情况下，循环都不是最高效的选择。之所以这么用，只是因为这里的数据量非常小，而且这样写最容易理解，当你写交叉验证代码的时候，请考虑到这一点。

循环结束之后，我们可以把尝试过的各个次数对应的多项式模型的效果画出来，如图6-6所示：

```
ggplot(performance, aes(x = Degree, y = RMSE, linetype = Data)) +
  geom_point() +
  geom_line()
```

从图6-6中可以清楚地看到，中间大小的次数对应的模型在测试数据上的表现最好。一方面，当次数过低，如1或2时，模型没有能拟合到数据的真正模式，在训练集和测试集的

效果都非常差。当一个模型过于简单，以致连训练数据都拟合得不够好时，称之为欠拟合（underfitting）。

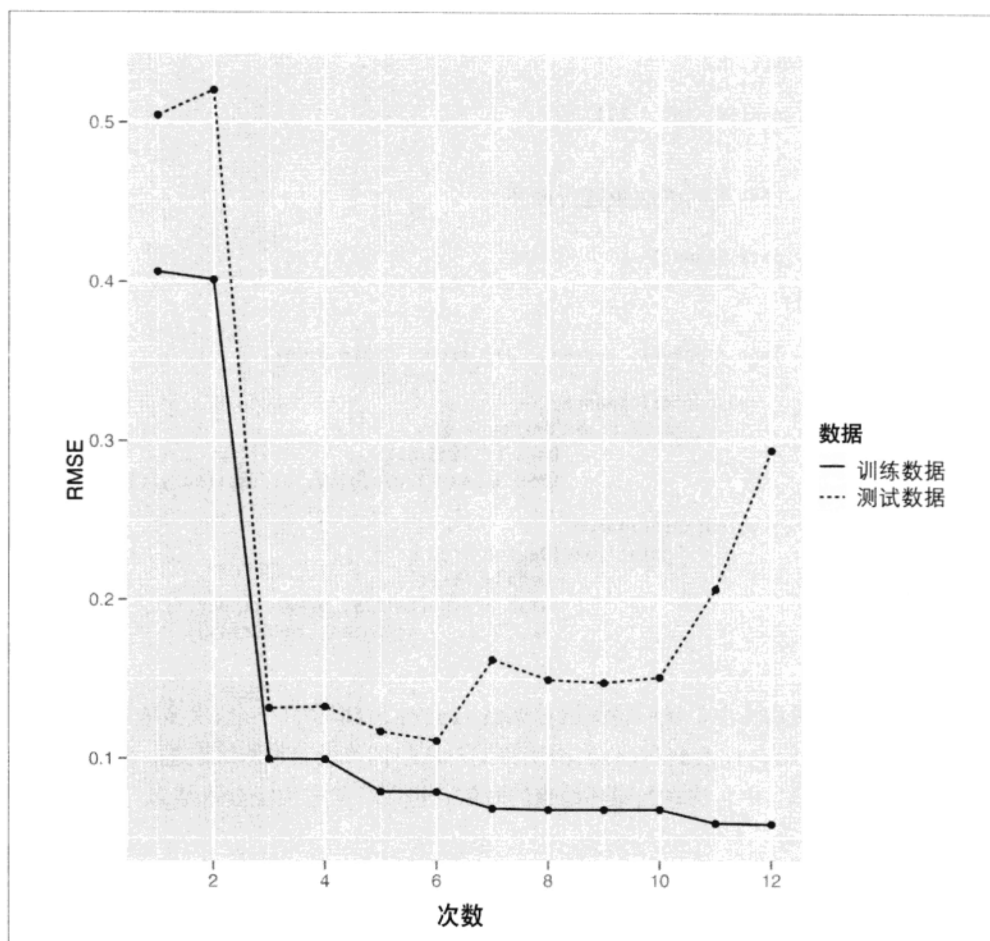


图6-6：交叉验证

另一方面，当次数太大，如11或12时，可以看到模型在测试数据上的表现又开始变差了。这是因为模型变得太复杂，拟合了在测试数据中并不存在的而在训练数据中存在的噪音。当模型开始拟合训练数据中的噪声时，就称之为发生了过拟合。可以从另一个角度理解过拟合：随着次数不断增加，训练误差和测试误差变化趋势开始不一致了——训练误差持续变小，而测试误差开始变大。模型对于任何它没见过的数据都没有泛化能力——这最终导致它过拟合。

幸运的是，从图中我们知道如何设置次数才能达到最好的效果。如果不使用交叉验证技

术，我们将很难发现这个既不欠拟合又不过拟合的微妙的中间点。

在简单介绍了如何通过交叉验证来避免过拟合之后，我们将开始讨论另外一种避免过拟合的技术——正则化。尽管我们可以使用交叉验证来证明正则化确实能够避免过拟合，但正则化与交叉验证在本质上是完全不同的。不过，我们还是会通过交叉验证来校正正则化算法，因此最终这两种方法又是紧密关联的。

使用正则化来避免过拟合

在本章中，我们一直说某个模型太复杂，但还没有给“模型复杂度”下一个正式的定义。如果要给多项式拟合的模型定义一个复杂度，我们可以说，多项式的次数越高它的复杂度越高。例如，一个2次多项式模型比一个1次多项式模型要复杂得多。

不过，这个定义对于都是一次多项式的线性回归并不适用。因此我们使用复杂度的另外一个定义：我们认为一个模型中特征的权重越大，这个模型越复杂。例如，我们可以认为模型`y ~ 5 * x + 2` 比 `y ~ 3 * x + 2`更复杂。同样，我们也认为`y ~ 1 * x^2 + 1 * x + 1`比`y ~ 1 * x + 1`更复杂。为了对这个定义有感性认识，我们会使用`lm`函数来拟合一个线性模型，通过累加返回的`coef`值，来度量它的复杂度：

```
lm.fit <- lm(y ~ x)
model.complexity <- sum(coef(lm.fit) ^ 2)
```

在这里我们累加的实际上是权重的平方，这是因为我们不希望累加时使正负权重互相抵消。这里取平方的步骤通常称作L2正则化（L2 norm）。另一种避免正负权重互相抵消的方式是累加它们的绝对值，这个方法称作L1正则化（L1 norm）。下面用R语言计算了L1和L2的结果：

```
lm.fit <- lm(y ~ x)
l2.model.complexity <- sum(coef(lm.fit) ^ 2)
l1.model.complexity <- sum(abs(coef(lm.fit)))
```

这里关于模型复杂度的定义可能看上去有点奇怪，不过一会儿你就会发现，它们确实能帮助我们避免过拟合。这是因为，复杂度的定义迫使我们在训练模型的过程中，让模型变得尽量简单。具体细节我们将在第7章讨论，这里的主要思想是，我们将要在“让模型尽量拟合训练数据”与“让模型尽量保持简单”之间做出权衡。这就是数据建模中最关键的一个抉择：因为我们要在数据的拟合效果和模型复杂度之间做出权衡，所以最终会在一个比较简单但是拟合得不够好的模型，和一个比较复杂但是拟合得非常好的模型之间做出选择。这个权衡，也就是我们所说的正则化，最终避免了模型去拟合那些训练数据中的噪声，进而避免过拟合。

现在，让我们看看在R语言中可以用来正则化的工具。在本章中，我们会使用`glmnet`程

序包，它提供了一个可以训练正则化的线性模型的函数：glmnet。为了了解glmnet函数是如何工作的，让我们再使用一次正弦波数据：

```
set.seed(1)

x <- seq(0, 1, by = 0.01)
y <- sin(2 * pi * x) + rnorm(length(x), 0, 0.1)
```

为了使用glmnet函数，首先把向量形式的x通过matrix函数转化成矩阵形式。然后，在调用glmnet函数时，需要注意这里的参数顺序，在lm函数中，y在~操作符前，而x在~之后；而在glmnet函数中正好相反：x是第一个参数，而y是第二个参数：

```
x <- matrix(x)

library('glmnet')
glmnet(x, y)

#Call: glmnet(x = x, y = y)
#
#      Df    %Dev    Lambda
# [1,] 0 0.00000 0.542800
# [2,] 1 0.09991 0.494600

# [3,] 1 0.18290 0.450700
# [4,] 1 0.25170 0.410600
# [5,] 1 0.30890 0.374200
...
#[51,] 1 0.58840 0.005182
#[52,] 1 0.58840 0.004721
#[53,] 1 0.58850 0.004302
#[54,] 1 0.58850 0.003920
#[55,] 1 0.58850 0.003571
```

以这种方式调用glmnet函数，将会得到回归模型所有可能的正则化结果。结果列表中最靠前的是glmnet函数执行最强正则化的结果，最靠后的是glmnet函数执行最弱正则化的结果。这里显示了结果的最前和最后各5条结果。

让我们解释一下这里展示的10行结果中每一列都是什么含义。这里每一行都包含了3列：1) Df；2) %Dev；3) lambda。第一列是Df，它指明模型中的非零权重有几个。但并不包括截距项，因为你肯定不想正则化它的大小。知道模型当中的非零权重个数非常有用，因为我们通常认为并不是所有的输入特征都是有用的，如果模型把某些输入特征的权重设为0，仍然有比较好的效果，那我们就更确认那些权重为0的输入特征是无关紧要的。当一个统计模型的大部分输入特征的权重都为0时，这个模型就是稀疏的（sparse）。当代机器学习研究领域的一个重要课题，就是开发能够得到稀疏统计模型的工具。

第二列是%Dev，其实就是模型的R^2值。第一行的0%是因为你把唯一的输入特征权重赋值

为0，所以这个模型实际上就是一个常数模型。最后一行的%Dev是59%，这个值和你直接使用lm函数时得到的R^2值是一样的，因为lm函数没有使用任何正则化。介于完全正则化和完全没有正则化的两个极端之间的那些模型的%Dev值介于9%~58%。

最后一列是lambda，对正则化来说，这是最重要的一个信息。lamdba是一个正则化算法的参数，这个参数控制了你准备拟合的模型的复杂度。它控制了你最终得到的模型参数，因此通常也称lambda为超参数（hyperparameter）。

我们将会在第7章中详细讨论lambda的含义，这里仅给出关于lambda的一个直观概念：lambda很大，说明你对模型的复杂度很在意，你对复杂的模型施以很大的“惩罚”，这个惩罚将迫使所有的模型权重趋向于0；反之，lambda很小，说明你对模型的复杂度漠不关心，你对复杂的模型几乎都不怎么“惩罚”。在极端情况下，我们可以把lambda设为0，那样将会得到一个完全没有正则化的线性回归模型，就像直接使用lm函数得到的模型一样。

不过，通常来说，为了得到一个最优的模型，我们设定的lambda是一个大小适中的值。那么如何得到这样一个最合适的lambda呢？这就是我们在正则化过程中需要用到交叉验证的地方了。但这一次，我们并不是对多项式回归的次数进行交叉验证，而是首先设定一个比较大的次数，比如10。然后使用不同的lambda分别在测试集上训练模型，再看它们在测试集上的效果如何。通过迭代多次不同的lambda，我们可以找到那个在测试集上效果最好的lambda。

警告： 你必须拿测试集来评估正则化的效果。因为随着正则化程度的增加，模型在训练集上的效果肯定是越来越差的，因此训练集上的效果原则上说没有任何意义。

按照前面介绍的方法，让我们来实际看一个例子。和以前一样，我们把数据拆分成一个训练集和一个测试集：

```
set.seed(1)

x <- seq(0, 1, by = 0.01)
y <- sin(2 * pi * x) + rnorm(length(x), 0, 0.1)

n <- length(x)

indices <- sort(sample(1:n, round(0.5 * n)))

training.x <- x[indices]
training.y <- y[indices]

test.x <- x[-indices]
test.y <- y[-indices]

df <- data.frame(X = x, Y = y)
```

```
training.df <- data.frame(X = training.x, Y = training.y)
test.df <- data.frame(X = test.x, Y = test.y)

rmse <- function(y, h)
{
  return(sqrt(mean((y - h) ^ 2)))
}
```

不过，这一次我们不是对次数进行循环，而是对lambda进行循环。幸运的是，我们不必每次都重新训练模型，因为只需一次调用，glmnet函数就已经保存了多个不同的Lambda和它们分别对应的模型。

```
library('glmnet')

glmnet.fit <- with(training.df, glmnet(poly(X, degree = 10), Y))

lambdas <- glmnet.fit$lambda

performance <- data.frame()

for (lambda in lambdas)
 {
  performance <- rbind(performance,
    data.frame(Lambda = lambda,
               RMSE = rmse(test.y, with(test.df, predict(glmnet.fit, poly(X,
                            degree = 10), s = lambda)))))
}
```

经过上述处理后，我们已经计算出了不同lambda对应的模型的效果，我们可以绘制一张直方图，如图6-7所示，从图中可以找到在测试集上效果最好的模型对应的lambda：

```
ggplot(performance, aes(x = Lambda, y = RMSE)) +
  geom_point() +
  geom_line()
```

从图 6-7中可以看出，当lambda接近0.05时，模型效果最好。因此我们可以用这个lambda在完整的数据集（包括训练集与测试集在一起的所有数据）上训练一个模型：

```
best.lambda <- with(performance, Lambda[which(RMSE == min(RMSE))])

glmnet.fit <- with(df, glmnet(poly(X, degree = 10), Y))
```

当我们在完整的数据集上训练出模型之后，可以通过coef来查看正则化的模型的结构：

```
coef(glmnet.fit, s = best.lambda)
#11 x 1 sparse Matrix of class "dgCMatrix"
#                    1
#(Intercept) 0.0101667
#1          -5.2132586
#2           0.0000000
#3           4.1759498
#4           0.0000000
```

```
#5           -0.4643476
#6            0.0000000
#7            0.0000000
#8            0.0000000
#9            0.0000000
#10           0.0000000
```

正如可以从这个结果表中看到的，虽然有10个特征可供选择，但实际上我们只使用了3个。这正是正则化背后的主要思想：宁愿选择像这样比较简单的模型，也不选择复杂的模型。有了正则化这个工具，我们可以采用一个较高次数的多项式进行拟合，也不用担心过拟合问题。

在熟悉了正则化的基本概念之后，让我们来看一下本章的案例研究。

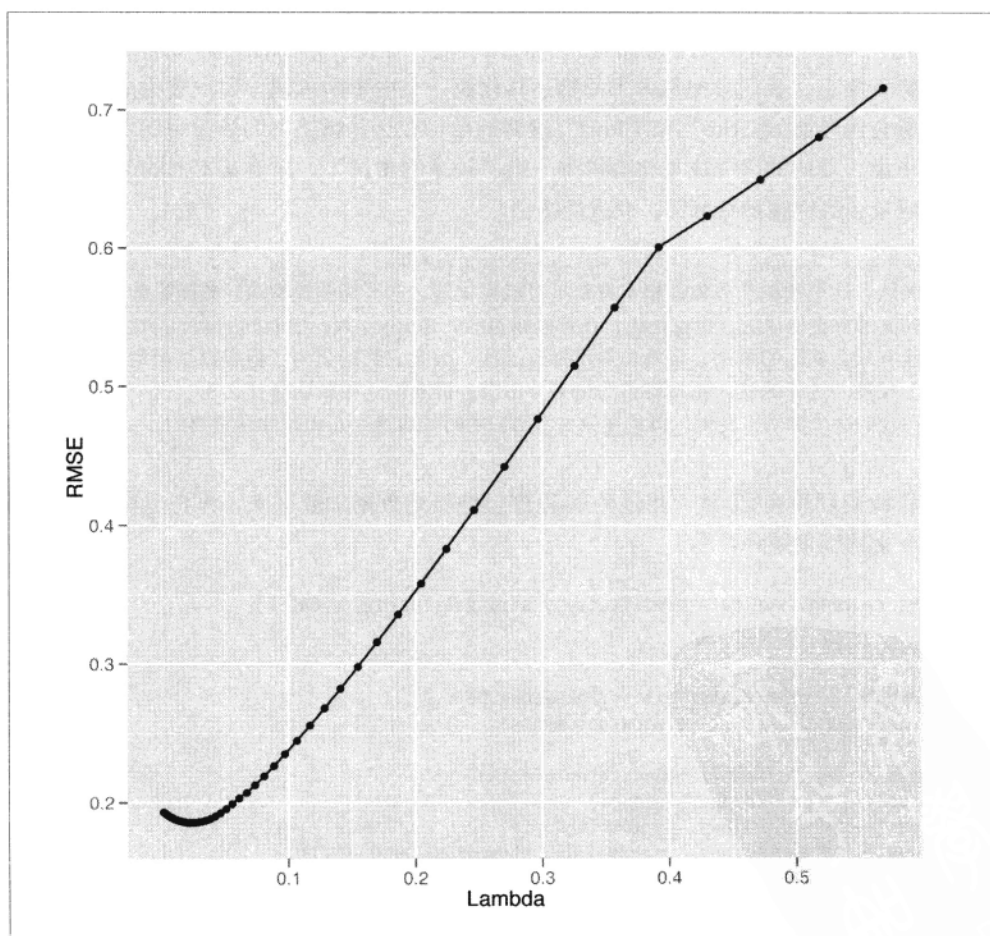


图6-7：不同lambda下的正则化

文本回归

交叉验证和正则化是两个非常强大的工具，可以使我们使用复杂的模型描述数据中隐藏的复杂模式，又不至于过拟合。应用正则化最有趣的案例之一，就是使用文本信息来预测一些连续的输出值。例如，我们可以根据一只股票的IPO招募书，来预测它股价的波动情况。当我们使用文本作为一个回归模型的输入时，输入特征个数（单词）几乎总是比样本数（文档）多。即使样本数比1-grams（单个词）多，我们还可以使用2-grams（2个词一组）或3-grams（3个词一组），所以最终，我们的n-grams还是会多过文档数的。因为数据集的列数比行数多，非正则化的线性回归总是会生成一个过拟合的模型。基于这个原因，为了得到有用的结果，我们必须使用一些正则化的手段。

为了便于理解，我们会找一个简单的案例来研究——根据O’Reilly出版社出版的销量前100的畅销书的封底描述文本来预测它们的相对流行程度。为了把这些文本描述转变成可用的输入集合，我们会把每本书的描述转化成一个由不同单词出现次数构成的向量，这个向量会保存诸如“the”和“Perl”这样的单词在这些畅销书的描述中出现的次数。从理论上说，我们的研究成果将会得到一组“神奇的单词”，只要某本书的封底描述里包含了尽量多的这些神奇单词，它就能畅销。

警告： 当然，这个预测任务也许根本就是不可能完成的。这可能是模型给一些随机的单词赋予了较高的权重造成的。也就是说，在那些畅销的O’Reilly出版社出版的书籍的描述当中，几乎没有什么相同的单词，但模型并不知道这些，因此它仍然会为了拟合模型而给一些单词附上权重。这样的话，模型得到的结果并不会告诉我们这些单词有什么用。这个问题未必会在我们这个案例中发生，但是在做文本回归的时候有这个意识是非常重要的。

首先，让我们利用在第3章使用过的tm程序包把原始数据加载进来，再把原始数据集转化成一个文档词项矩阵：

```
ranks <- read.csv('data/oreilly.csv', stringsAsFactors = FALSE)

library('tm')

documents <- data.frame(Text = ranks$Long.Desc.)
row.names(documents) <- 1:nrow(documents)

corpus <- Corpus(DataframeSource(documents))
corpus <- tm_map(corpus, tolower)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, removeWords, stopwords('english'))

dtm <- DocumentTermMatrix(corpus)
```

在这里，我们首先把CSV文件中的数据加载到ranks变量中，创建一个tm可以理解的、其

中包含了书籍描述的数据框，然后基于这个数据框创建一个语料库，将文本中的单词全部统一成小写，去掉多余的空格，删掉英语中最常见的停用词，然后创建我们的文档词项矩阵。做完上面这些工作，数据预处理的工作就算完成了。接下来，我们要将变量做一些小小的操作变换，来把回归问题转化成glmnet函数可以处理的形式：

```
x <- as.matrix(dtm)
y <- rev(1:100)
```

在这里，我们把文档词项矩阵转变成一个简单的数值矩阵，以方便后续处理。同时，我们把预测的y值和排名颠倒过来对应一下——这样排名第一个书的y值是100，而排名第100的书的y值是1。我们这么做是为了让那些能预测书籍流行度的单词的权重是正的。如果我们直接使用原始的排名而不逆序处理，那些能预测书籍流行度的单词的权重将会是负的。我们觉得这不那么直观，当然，这两种编码方式在本质上并没有差别。

最后，在运行回归分析之前，我们初始化随机种子，并且加载glmnet程序包：

```
set.seed(1)
library('glmnet')
```

初始化工作结束之后，我们可以对几个可选的lambda进行循环，来看一下哪一个lambda能在测试集上得到最好的效果。因为数据并不多，所以对于每一个lambda，我们执行50次不同的数据拆分，以提高我们评估正则化效果的准确程度。在下面的代码中，我们首先为lambda设定一个值，然后将数据拆分成不同训练集和测试集50次，再分别评估每一次拆分的效果。

```
performance <- data.frame()

for (lambda in c(0.1, 0.25, 0.5, 1, 2, 5))
{
  for (i in 1:50)
  {
    indices <- sample(1:100, 80)
    training.x <- x[indices, ]
    training.y <- y[indices]

    test.x <- x[-indices, ]
    test.y <- y[-indices]

    glm.fit <- glmnet(training.x, training.y)
    predicted.y <- predict(glm.fit, test.x, s = lambda)
    rmse <- sqrt(mean((predicted.y - test.y) ^ 2))

    performance <- rbind(performance,
                         data.frame(Lambda = lambda,
                                    Iteration = i,
                                    RMSE = rmse))
  }
}
```

所有这些lambda对应的模型的效果都计算完毕，我们就可以比较出哪一个模型表现最好：

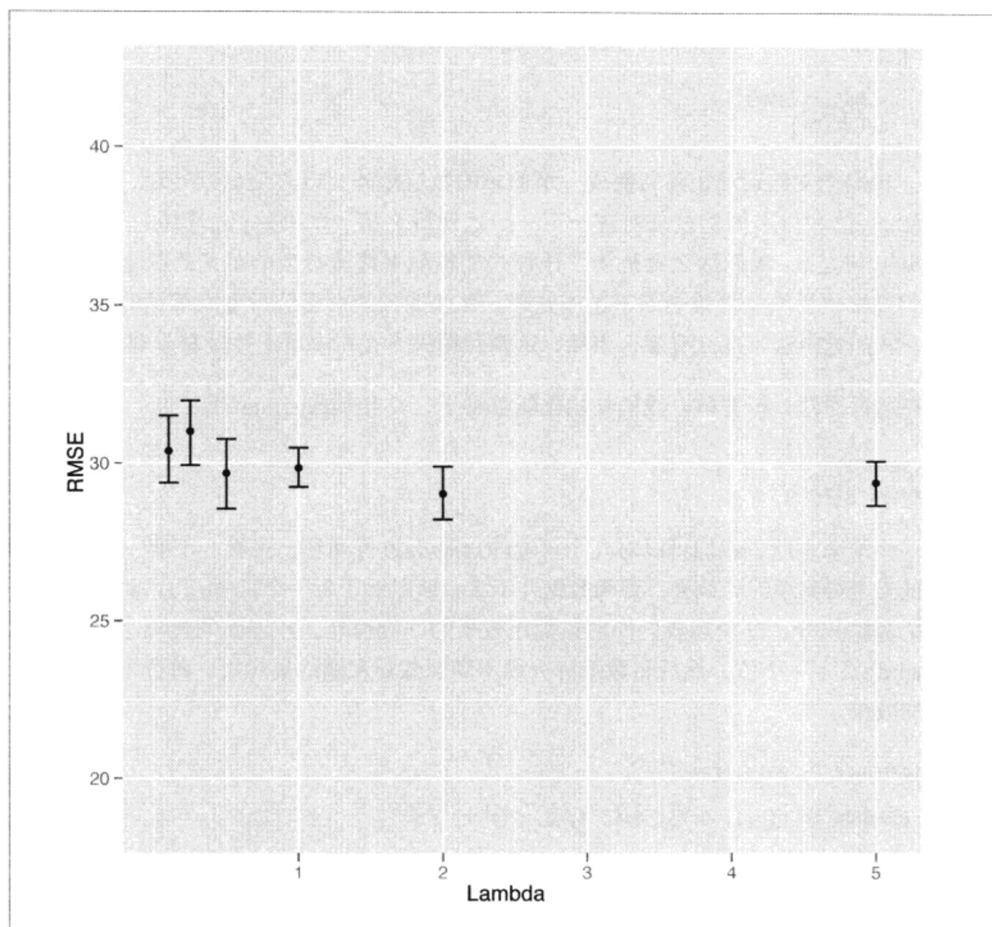


图6-8：O'Reilly书籍销量预测模型在不同lambda下的结果

```
ggplot(performance, aes(x = Lambda, y = RMSE)) +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'errorbar') +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'point')
```

遗憾的是，从图6-8中我们发现，我们为数据建立一个统计模型的尝试失败了，这是本书中这么多次尝试中的第一次。显然，随着lambda越来越大，模型的表现越来越好——但这种情况只有在模型简化到了一个常数模型时才会发生。这说明它根本没有使用任何文本信息。简而言之，我们的文本回归模型没有发现任何有意义的信息。当应用模型到测试集上时，我们发现，它给出的预测完全是随机噪声。

尽管这意味着期望中的能保证我们的书畅销的“神奇的单词”并不存在，但这对于任何一个在机器学习领域工作和学习的人来说，都是需要学习的重要一课：有时候，我们在数据中找不到任何模式。正如John Tukey所说：

> 数据中也许并没有答案。有一堆数据和对答案的热切渴望，并不能确保真的能从这堆数据中提取出合理的预期答案。

逻辑回归来帮忙

但我们手头的这份数据也不致一无是处。虽然不能创建一个从文本预测排名的工具，但我们也许可以尝试一些简单的，比如说预测一下某本书能否进入销量排名前50名。

为了做到这一点，我们要把这个回归问题转变成一个分类问题。我们不再预测从1到无穷大的排名，转而做一个简单的二分判断：这本书能否进入销量前50名？

这个二分判断非常简单明了，因此我们有理由相信能从这个较小的数据集中得到一些有用的信息。首先，让我们为数据集增加一个类别标签：

```
y <- rep(c(1, 0), each = 50)
```

在这里，我们使用了在第2章使用过的0/1虚拟变量编码，如果一本书在销量排行榜前50名就用1表示，否则用0表示。采用这样的虚拟变量，我们就可以使用在第2章结尾介绍的逻辑回归模型分类算法来预测一本书能否出现在销量排行榜的前50名。

逻辑回归，本质上是一种回归，它预测的其实是一个样本属于两个类别之中某一个的概率值。因为概率值总是介于0～1，所以我们可以以0.5作阈值来创建一个分类算法。除了输出介于0～1以外，逻辑回归本质上和线性回归是一致的。唯一的区别是你需要根据输出是否高于阈值来做出一个分类判断。首先我们会向你演示一下，在R语言中使用逻辑回归是多么简单，然后再来完成最后的阈值判断的过程。

首先，我们要用全体数据来拟合一个逻辑回归模型，这很简单：

```
regularized.fit <- glmnet(x, y, family = 'binomial')
```

与前面为了做线性回归而调用`glmnet`函数的唯一区别在于，在这里我们调用`glmnet`时增加了一个额外的`family`参数，这个参数控制了预测的误差类型。我们在第5章没有详细讨论误差族，线性回归假定误差服从高斯分布，而逻辑回归模型假定误差服从二项分布（binomially distributed）。我们并不会讨论二项分布的细节，不过你应该知道这是掷硬币时会得到的一个分布。基于这个原因，二项分布产生的误差是0或1，当然，就是分类问题需要使用的误差族。

为了详细说明，我们看一下调用glmnet函数的三种方式：

```
regularized.fit <- glmnet(x, y)

regularized.fit <- glmnet(x, y, family = 'gaussian')

regularized.fit <- glmnet(x, y, family = 'binomial')
```

第一种调用方式，是我们之前做线性回归时使用的。第二种调用方式与第一种调用方式完全等价，只是我们显式地指明了family的默认参数：gaussian。第三种调用方式是我们采用逻辑回归时使用的调用方式。正如你所看到的，从线性回归到逻辑回归，你所需要做的，仅仅是改变误差族参数[注2]。

从全体数据拟合出一个逻辑回归模型之后，让我们看一下对模型使用predict函数会得到什么样的预测结果：

```
predict(regularized.fit, newx = x, s = 0.001)
#1   4.884576
#2   6.281354
#3   4.892129
...
#98  -5.958003

#99  -5.677161
#100 -4.956271
```

如你所见，输出结果包含了正数和负数，而并不是我们所期望的0或1。我们有两种办法来处理原始的输出。第一个方法是以0为阈值，使用ifelse函数来做出0/1的预测：

```
ifelse(predict(regularized.fit, newx = x, s = 0.001) > 0, 1, 0)

#1   1
#2   1
#3   1
...
#98  0
#99  0
#100 0
```

第二种方法是把原始预测输出转换成我们更容易理解的概率值，这样我们还是需要以0.5为阈值来给出和前面一样的0/1预测。为了把逻辑回归的原始预测转换成概率值，我们需要使用boot程序包的inv.logit函数：

```
library('boot')
inv.logit(predict(regularized.fit, newx = x, s = 0.001))
```

注2：其实还可以使用很多其他的误差族，不过要讲清楚这部分内容需要整整一本书，因此我们鼓励读者阅读相关资料！

```
#1   0.992494427
#2   0.998132627
#3   0.992550485
...
#98  0.002578403
#99  0.003411583
#100 0.006989922
```

如果你想知道`inv.logit`的细节，我们建议你去看一下这个函数的源代码，以便搞清楚这里面数学运算的细节。针对本案例的目的来说，我们仅仅希望你能了解，逻辑回归模型的输出需要经过逆logit函数转换才能变成概率输出。基于这个原因，逻辑回归也称为`logit`模型。

无论使用哪种转换方式将逻辑回归的输出转换成0/1，这都没有关系，重要的是逻辑回归为你提供了一个分类工具，这个工具可以让你像在第5章使用线性回归那样容易地进行分类。那么，让我们来看一下，使用逻辑回归来预测书籍能否进入畅销排行榜前50名的效果怎样。可以发现，这里的实现代码与前面用来预测排名的线性回归的代码相差不大：

```
set.seed(1)

performance <- data.frame()

for (i in 1:250)
{
  indices <- sample(1:100, 80)
  training.x <- x[indices, ]
  training.y <- y[indices]

  test.x <- x[-indices, ]
  test.y <- y[-indices]

  for (lambda in c(0.0001, 0.001, 0.0025, 0.005, 0.01, 0.025, 0.5, 0.1))
  {
     glm.fit <- glmnet(training.x, training.y, family = 'binomial')
     predicted.y <- ifelse(predict(glm.fit, test.x, s = lambda) > 0, 1, 0)
     error.rate <- mean(predicted.y != test.y)

     performance <- rbind(performance,
                          data.frame(Lambda = lambda,
                                     Iteration = i,
                                     ErrorRate = error.rate))
  }
}
```

这一部分代码与前面使用过的线性回归代码的差异有如下几个方面：1）对于`glmnet`函数的调用，在逻辑回归中，我们使用的是二项分布的误差族参数；2）对逻辑回归的原始输出进行阈值转换后变成0/1预测结果；3）使用错误率而不是RMSE来度量模型的效

果。另外一个你可能已经注意到的差别在于，我们执行拆分的循环次数是250次而不是50次，我们这么做的目的是为了对不同lambda对应的错误率有更准确的估计。因为我们发现错误率最终变得非常接近50%，而我们希望确定模型的预测结果比随机猜一个的结果要好。为了让循环更高效，我们交换了lambda循环和拆分循环次序，这样我们就不必为每一个lambda重新做多次拆分了。这节省了不少时间，这提醒我们，编写高效的机器学习代码需要你像一个优秀的程序员那样有高效的编码习惯。

我们对效果比较感兴趣，接下来把不同的lambda对应的错误率画在直方图上，如图6-9所示：

```
ggplot(performance, aes(x = Lambda, y = ErrorRate)) +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'errorbar') +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'point') +
  scale_x_log10()
```

结果告诉我们，把回归改成分类的尝试还是比较成功的。使用较小的lambda，我们预测一本书能否进入热销榜排名前50的效果比随机猜要好，这很令人欣慰。看来，这份数据尽管还不足以让我们拟合出一个复杂的可以预测排名的工具，但是却足够让我们拟合出一个比较简单的，仅仅预测书籍能否进入热销榜前50名的分类器。

让我们以这样一个普适性经验来结束这一章：有时候，简单的反而更好。为了在测试数据上有更好的效果，正则化迫使我们选择简单的模型。将回归模型转变成分类模型通常会让你得到更好的效果，因为有时候你手头上的资源（数据等）足够训练出一个简单的二分分类器，却不够训练出一个复杂的直接预测排名的工具。

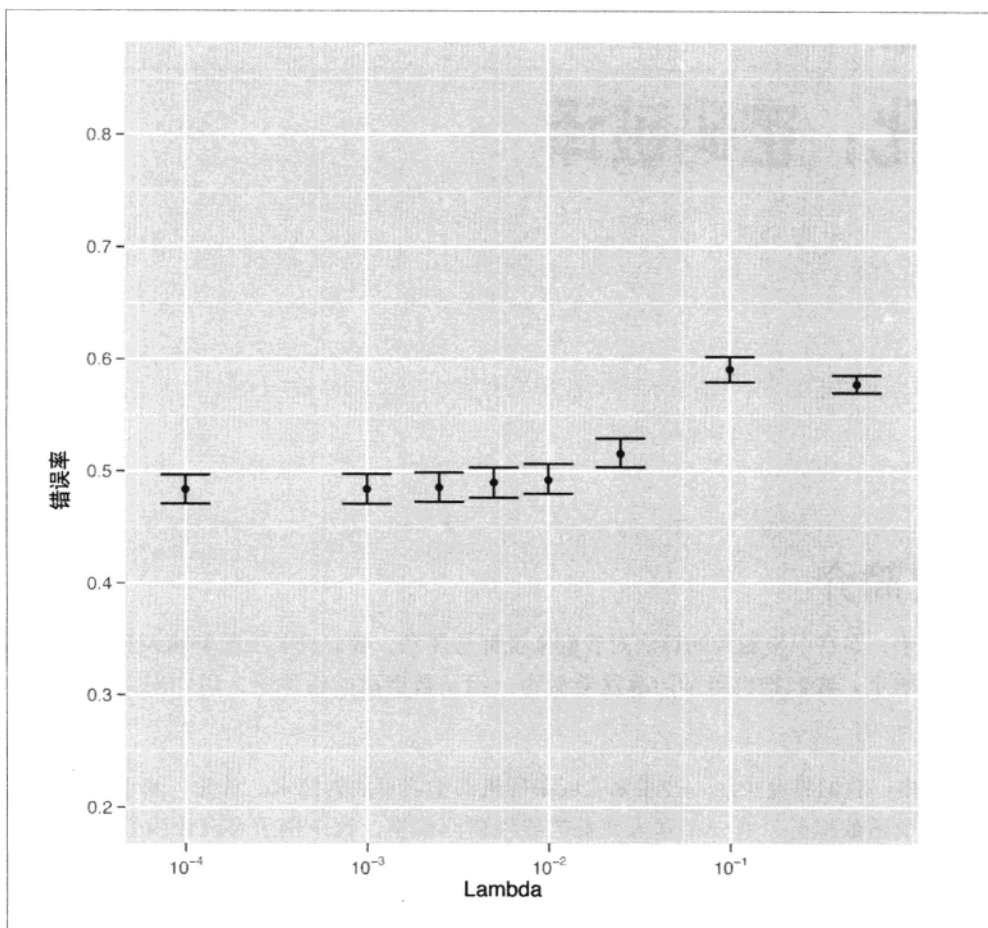


图 6-9：不同的lambda对应的书籍是否在热销榜前50名的错误率

第7章

优化：密码破译

优化简介

目前为止，本书中接触到的算法对我们来说都是黑盒，我们只关注理解输入和输出是什么。本质上，我们把机器学习算法看做由一组函数组成的软件库，用它来完成预测任务。

在本章中，我们将会关注一些用来实现基础机器学习算法的技术。首先，基于前面的知识写一个函数拟合只有一个输入变量的线性回归模型。这个例子可以让我们把从数据中拟合一个模型看成是一个优化问题。我们可以这样来理解优化问题：假设有一台机器，机器上有一些把手，我们可以操作这些把手来对机器进行不同的设置，然后可以衡量这台机器在当前设置下运行的效果如何。我们希望找到对这台机器的一种最佳设置，在某些简单的度量标准下，使得这台机器运行的效果最优。这个最佳设置称为最优点（optimum）。达到最优点的过程称为优化（optimization）。

在理解了优化的原理之后，我们会着手处理本章的主要工作：构建一个简单的密码破译系统，它把解密一串 密文当做一个优化问题。

因为需要写一个线性回归函数，所以再次使用前面章节中的身高和体重数据来举例。正如之前做过的那样，假如知道了一个人的身高，我们就可以通过一个函数来计算其体重。特别地，假定这是一个线性函数，用R语言描述如下所示：

```
height.to.weight <- function(height, a, b)
{
  return(a + b * height)
}
```

在第5章中，我们详细讨论了如何使用lm函数来计算这个线性函数的斜率和截距。在上面这个案例中，变量b是斜率，而变量a是截距，相当于一个身高是0的人的体重应该是多少。

有了这个函数之后，我们怎么知道a和b取什么值最好呢？这就是我们使用优化的地方：首先要为这个根据身高来预测体重的函数的效果定义一个度量标准，然后改变a和b的值，尝试来优化函数的效果，直到函数的效果不能再改进为止。

我们该怎么做到这一切？其实lm函数已经都替我们做完了。它会首先优化一个简单的误差函数，然后通过一种非常特殊的算法找到最优的a和b，当然这个特定的算法仅适用于普通的线性回归。

让我们先看看lm函数得到的a和b的值：

```
heights.weights <- read.csv('data/01_heights_weights_genders.csv')

coef(lm(Weight ~ Height, data = heights.weights))
#(Intercept) Height
#-350.737192 7.717288
```

为什么这样的a和b的值是合理的？为了回答这个问题，我们需要知道lm函数使用的是什么误差函数。我们在第5章曾简单提到，lm函数是基于平方误差的，它的工作原理如下：

1. 选定a和b；
2. 输入一个身高，预测对应的体重；
3. 误差=真实的体重-预测的体重；
4. 将误差平方；
5. 累加所有的训练样本的平方误差。

注意： 为了便于理解，我们通常对累积误差取平均，再开方处理。不过，对于优化而言，这没必要，只计算累加的平方误差可以节省一点点运算时间。

最后两步是紧密相关的：如果我们不把误差累加起来，对误差取平方没有意义，而如果我们不对误差进行平方，而直接将第3步得到的原始误差累加起来，最终正负误差会互相抵消而为0。

注意： 要证明这一点并不难，不过这需要一些代数运算，这并不是本书准备讨论的话题。

让我们来看一下具体的实现代码：

```
squared.error <- function(heights.weights, a, b)
{
  predictions <- with(heights.weights, height.to.weight(Height, a, b))
  errors <- with(heights.weights, Weight - predictions)
  return(sum(errors ^ 2))
}
```

为了对这个过程有一个感性认识，下面把特定a和b值对应的squared.error计算出来，如表7-1所示：

```
for (a in seq(-1, 1, by = 1))
{
   for (b in seq(-1, 1, by = 1))
   {
      print(squared.error(heights.weights, a, b))
   }
}
```

表7-1：一组a和b的值对应的平方误差

a	b	平方误差
-1	-1	536271759
-1	0	274177183
-1	1	100471706
0	-1	531705601
0	0	270938376
0	1	98560250
1	-1	527159442
1	0	267719569
1	1	96668794

正如你看到的，某些a和b对应的squared.error比其他的a和b要小得多。这表明，我们现在已经有一个合适的函数来定义什么是a和b的“最佳”值。这是优化问题的第一步：找到一个我们想要最大化或最小化的度量标准。这个度量标准通常称为目标函数（objective function）。接下来优化问题就变成了寻找最合适的a和b，使得目标函数尽可能小或者尽可能大。

一个最显而易见的方法称为网格搜索（grid search）：为不同的a和b创建一个像我们刚刚展示的那样的表格，为表格中所有的a和b计算对应的squared.error，然后选择最小的squared.error对应的a和b的值。因为这种方法保证在你列举的表格中，总是能找到最佳的a和b，所以这种方法也算合理。不过，它有一些非常严重的问题：

- 该如何选择表格中各个变量之间的间隔？a选择0、1、2、3合适吗？或者b选择0、0.001、0.002、0.003合适吗？换句话说，搜索的步长是多少？要回答这个问题，需

要同时评估确定哪种选择的信息量更大，这个评估过程的计算量非常大，事实上还需要对搜索步长进行优化，这样就引入了另一个优化问题。如此下去，将陷入无限的循环。

- 如果你想要搭建一张包含2个变量，每个变量有10个可选值的表格，那么这张表格需要有100行。不过，如果你想评估10个变量，每个变量有100个可选值，那么你需要一张行数为10的100次方的表格。这种问题规模呈指数增长的情况在机器学习领域非常普遍，称为维度灾难（Curse of Dimensionality）。

如果想对上百甚至上千的输入特征进行线性回归，网格搜索就不是一种合适的优化算法。那么我们该怎么办？幸运的是，计算机科学家和数学家们研究优化问题已经很长时间了，并且实现了一批现实可行的优化算法。在R语言中，如果遇到一个优化问题，通常第一选择是尝试optim函数，这个函数封装了大多数主流的优化算法。

为了展示如何使用optim函数，我们将使用它来拟合线性回归模型。我们希望通过optim函数得到的a和b的值与我们直接调用lm函数得到的值不会相差太多：

```
optim(c(0, 0),
      function (x)
      {
        squared.error(heights.weights, x[1], x[2])
      })
#$par
#[1] -350.786736 7.718158
#
#$value
#[1] 1492936
#
#$counts
#function gradient
#     111       NA
#
#$convergence
#[1] 0
#
#$message
#NULL
```

如上例所示，optim函数需要一些额外的参数。首先，你需要把打算优化的参数封装在一个数值向量里，并将其当做第一个参数传递给optim函数。在这个例子中，我们要优化的是a和b，默认的初始数值向量就应该是c(0, 0)。接下来，你需要传递的第二个参数是一个函数，这个函数接受一个数值向量（例子中的x）作为输入，这个数值向量包含了你想要优化的目标变量。因为通常写的函数都包含不同的名字的多个参数，因此需要将误差函数封装到一个只接受一个参数的匿名函数里。在这个例子里，你可以看到是如何封装squared.error函数的。

运行optim函数，我们会分别得到a和b的值，它们保存在par变量里面。我们看到a和b的值和直接通过lm函数得到的值非常接近，这说明optim函数是正确的[注1]。实际上，lm函数使用了专门适用于线性回归的算法[译注1]，因此它得到的值比optim函数更精确一些。不过，如果你打算解决的优化问题并不是线性回归，那么optim函数还是很好的选择。

接下来了解一下从optim函数得到的其他返回值：第一个是value函数，这个值告诉我们在optim函数返回的最优点时，平方误差值具体是多少。第二个是counts，它告诉我们optim函数分别执行了输入函数（名为function）以及梯度（gradient）多少次。

注意： 如果你不明白“梯度”是什么意思，没关系。由于我们不喜欢手动计算梯度，因此通常都把梯度计算的工作留给optim函数自己去做，而不特别指定任何的梯度参数。到目前为止，一切还好，不过你的情况可能不一样。

第三个是convergence，它告诉我们optim函数是否足够有把握找到最优点。如果一切没问题，那么它的值应该是0。而当它非0时对应的错误信息可以在optim函数的帮助文档中找到。最后，message变量保存了任何可能对我们有用的其他信息。

总而言之，optim函数基于一些微积分的知识帮助我们完成了优化的过程。因为它依赖非常复杂的数学推导，所以我们并不去研究它内部究竟是怎么做到的。不过，我们可以用图表的方式非常直观地展示一下optim函数是怎么做到的。假定你首先确定b为0，然后想找到最优的a。你可以用下面的代码来计算平方误差：

```
a.error <- function(a)
{
 return(squared.error(heights.weights, a, 0))
}
```

为了找出最优的a，你可以把平方误差看成是a的函数，并把它们的函数关系使用curve函数画出来，curve函数能够计算一个函数或表达式在不同输入下的对应输出值，然后把不同输入和对应输出值画出来。在下面的例子里，针对不同的x值计算a.error，鉴于R语言在表达式计算时的怪异语法，我们需要使用sapply函数来帮忙。

```
curve(sapply(x, function (a) {a.error(a)}), from = -1000, to = 1000)
```

从图7-1来看，似乎有一个单独的a值是最优的，远离a的其他任何点的平方误差都会变得更大。当情况如此时，我们认为存在一个全局最优点（global optimum）。在这样的情况下，optim函数一旦计算了某一个a对应的误差函数值，就可以利用形状信息知道它下

注1：　或者至少它和lm函数一样糟糕。

译注1：此处是指最小二乘法。

一步应该往哪个方向找寻下一个更优的a。optim函数利用这个局部信息来学习你的问题的整体结构，因此它可以非常快速地找到最优点。

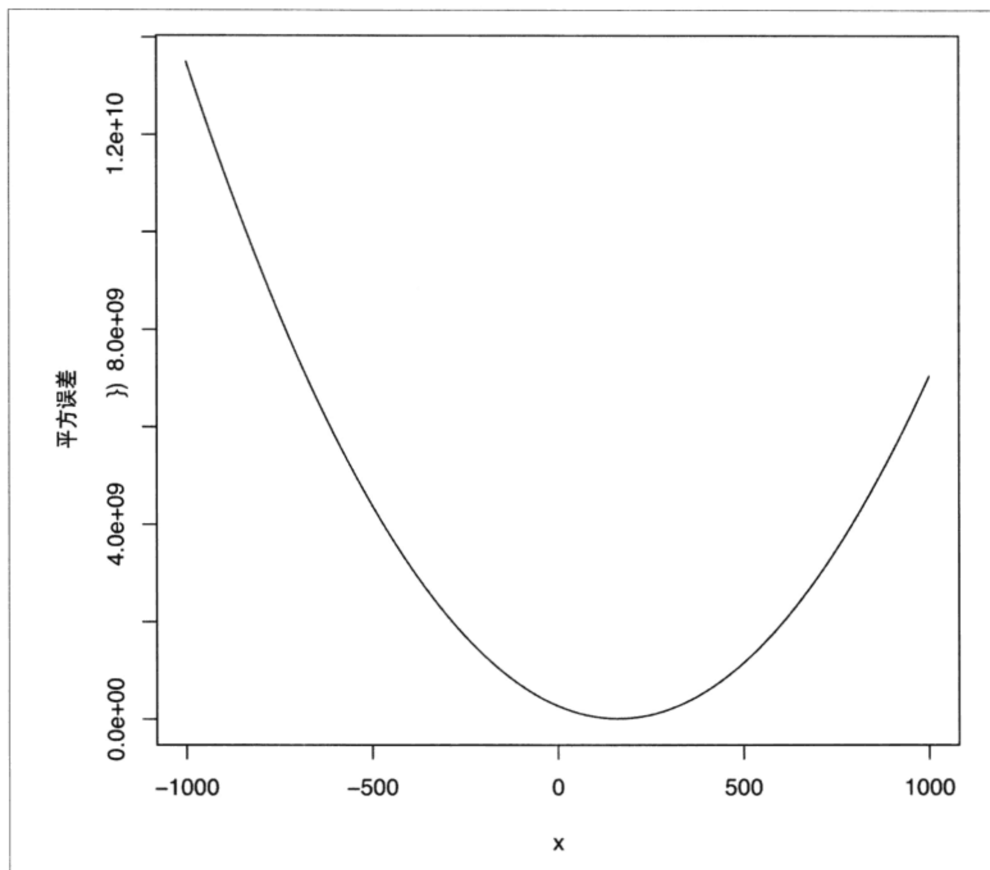


图7-1：改变a时的不同平方误差

为了对整个回归问题有更好的理解，我们同样也要看一下，改变b的时候误差函数是怎么变化的：

```
b.error <- function(b)
{
   return(squared.error(heights.weights, 0, b))
}

curve(sapply(x, function (b) {b.error(b)}), from = -1000, to = 1000)
```

从图7-2来看，似乎误差函数对b来说也有一个全局最优点。考虑到前面我们发现对于a也有一个全局最优点这一事实，这意味着optim函数应该可以在全局找到最优的a和b，来最小化我们的误差函数。

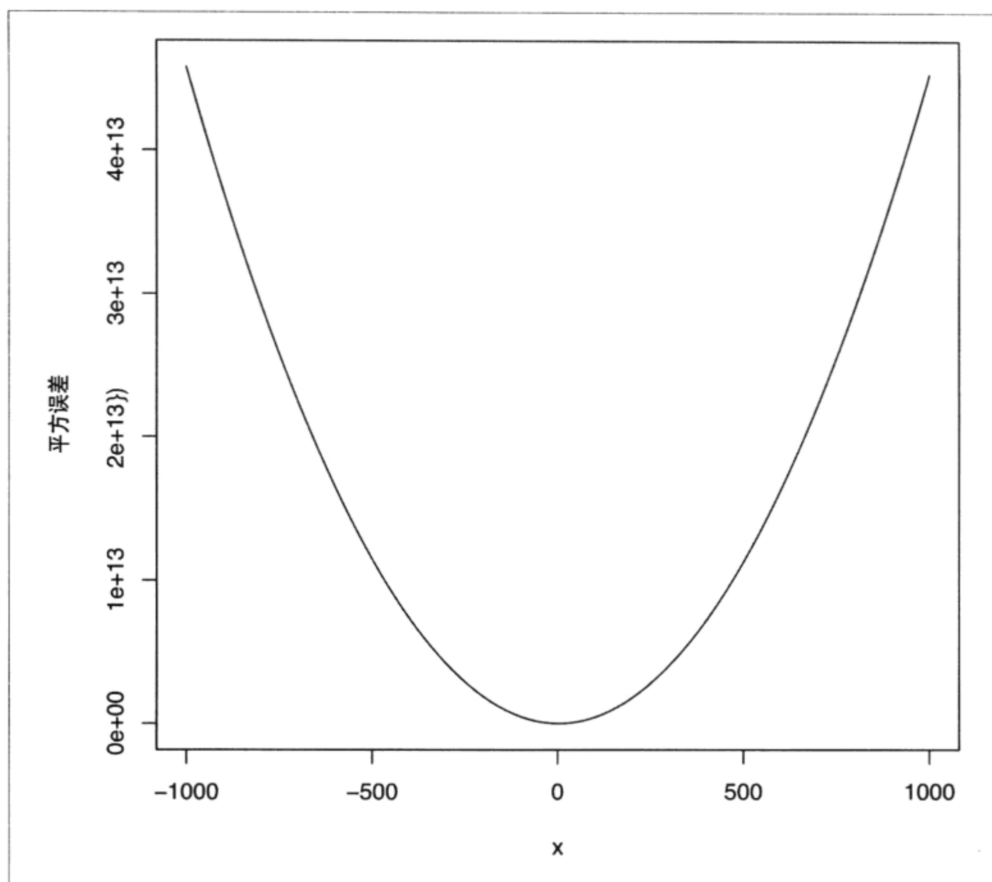


图7-2：改变b时的不同平方误差

一般来说，我们可以认为，基于微积分的数学原理，由于optim函数可以同时对所有的变量一起优化，因此optim函数才能够达到优化的目的。之所以它比网格搜索要快，是因为它可以利用当前点的信息来推断出当前点周围点的信息。这让它知道接下来它应该向哪个方向去寻找最优点。正是这种自适应性的工作方式使得它比网格搜索要高效得多。

岭回归

对于如何使用optim函数，我们算是入门了，接下来可以使用优化算法来实现自己版本的岭回归（ridge regression）了。岭回归是一种特殊的回归，其中包含了正则化，正如我们在第6章所说的。岭回归与普通的最小二乘回归算法的唯一区别在于：岭回归把回归系数的本身当做误差项的一部分，这促使回归系数变小。在本案例中，会倾向于选择更接近0的斜率和截距。

除了改变误差函数以外，岭回归增加的唯一复杂度在于，我们需要引入一个额外的参数lambda，这个参数用来权衡我们是希望有较小的平方误差，还是希望有较小的a和b以避免过拟合。这个额外引入正则化算法里面的参数称为超参数，在第6章曾经简单介绍过。一旦选定了lambda，可以写出如下所示的岭回归误差函数：

```
ridge.error <- function(heights.weights, a, b, lambda)
{
  predictions <- with(heights.weights, height.to.weight(Height, a, b))
  errors <- with(heights.weights, Weight - predictions)
  return(sum(errors ^ 2) + lambda * (a ^ 2 + b ^ 2))
}
```

正如在第6章所说的，我们可以使用交叉验证来选择最合适的lambda。在本章余下的内容中，我们假设你已经这么做了，并且假设选定的最优的lambda是1。

当我们定义好了岭回归的误差函数时，调用optim函数来求解岭回归问题，就如同求解普通的最小二乘问题一样简单了：

```
lambda <- 1

optim(c(0, 0),
      function (x)
      {
        ridge.error(heights.weights, x[1], x[2], lambda)
      })
#$par
#[1] -340.434108   7.562524
#
#$value
#[1] 1612443
#
#$counts
#function gradient
#     115       NA
#
#$convergence
#[1] 0
#
#$message
#NULL
```

看一下optim函数的输出就会发现，得到的线性函数的截距是-360，斜率是7.7，都比直接使用lm函数得到的结果略微缩小了一些。在这个仅仅作为示意的例子中，这没有什么太大的意义，不过在第6章中运行的大规模回归中，为较大的回归系数增加一个惩罚项，会帮助我们得到更有意义的结果。

除了观察拟合出来的系数以外，还可以重复前面调用过的curve函数，观察为什么optim函数对岭回归也适用。结果如图7-3和图7-4所示。

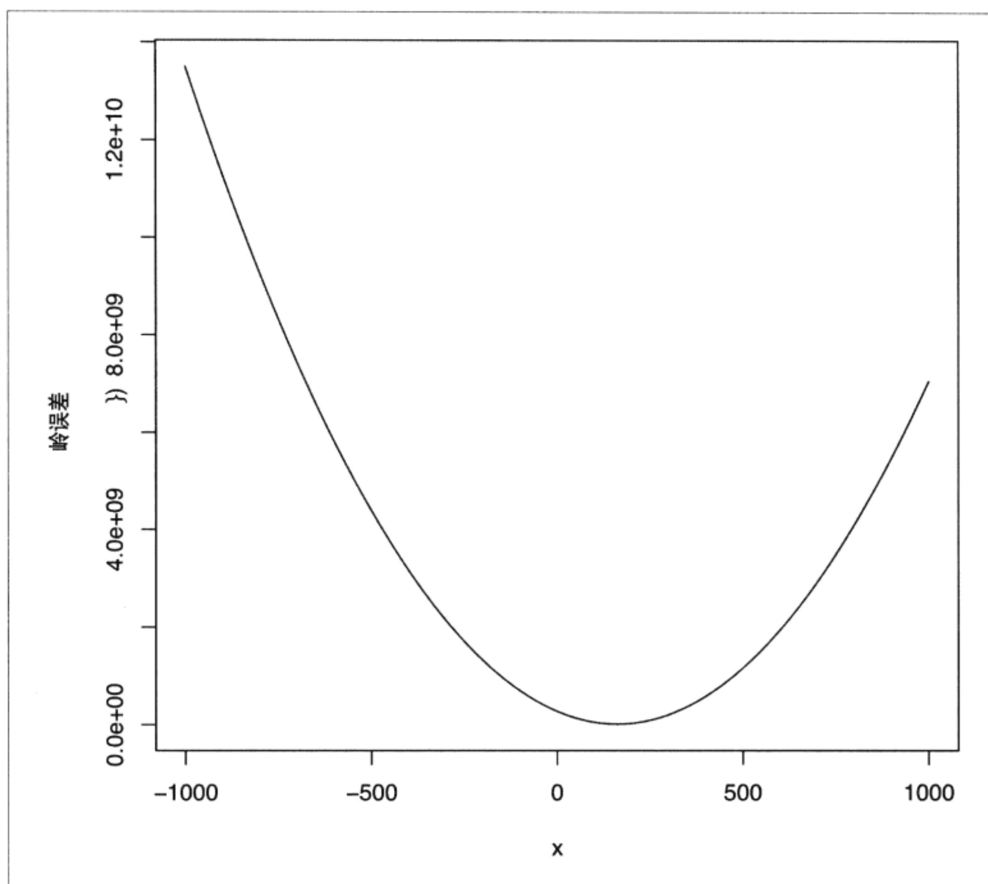


图7-3：改变a时的岭误差值

```
a.ridge.error <- function(a, lambda)
{
  return(ridge.error(heights.weights, a, 0, lambda))
}

curve(sapply(x, function (a) {a.ridge.error(a, lambda)}), from = -1000, to = 1000)

b.ridge.error <- function(b, lambda)
{
  return(ridge.error(heights.weights, 0, b, lambda))
}

curve(sapply(x, function (b) {b.ridge.error(b, lambda)}), from = -1000, to = 1000)
```

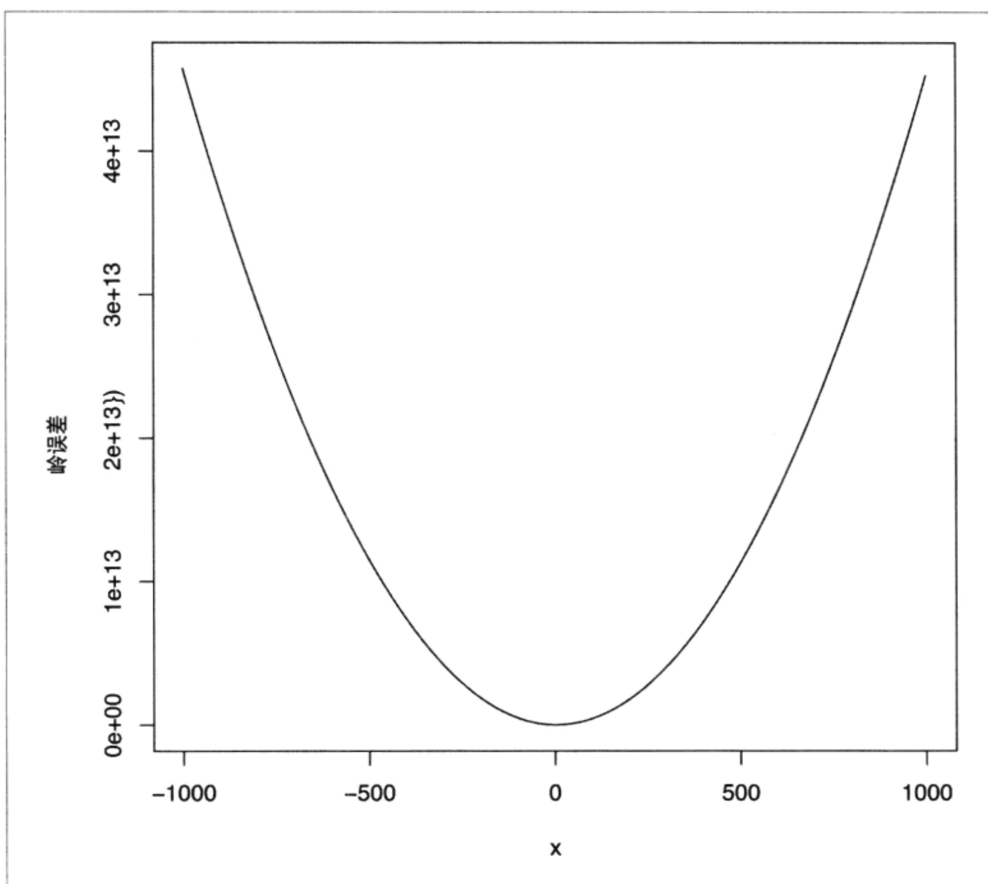


图7-4：改变b时的岭误差值

但愿这个例子能够让你相信，只要理解了如何使用optim函数来最小化一些预测误差，就可以明白很多机器学习问题。我们非常鼓励你在自己的例子上应用optim函数，并且尝试一下自己“发明”的各种不同的误差函数。这对你非常有用，特别是，当你尝试使用下面的绝对值误差函数时：

```
absolute.error <- function
(heights.weights, a, b)
{
  predictions <- with(heights.weights, height.to.weight(Height, a, b))
  errors <- with(heights.weights, Weight - predictions)
  return(sum(abs(errors)))
}
```

根据微积分的相关原理可知，这个误差函数在optim函数里工作得不那么好。如果没有扎

实的微积分基础，可能很难完全理解这里的原因，不过还是可以使用curve函数，以可视化的方法来对原因稍加阐释的：

```
a.absolute.error <- function(a)
{
  return(absolute.error(heights.weights, a, 0))
}
curve(sapply(x, function (a) {a.absolute.error(a)}), from = -1000, to = 1000)
```

正如你在图7-5中所看到的那样，绝对值误差曲线要比平方误差或者岭误差曲线锋利的多。正因为它太锋利了，以致optim函数无法通过一个单独的点来获知它该向哪个方向前进的信息，进而无法达到全局最优，尽管我们可以很清楚看到确实存在一个全局最优点。

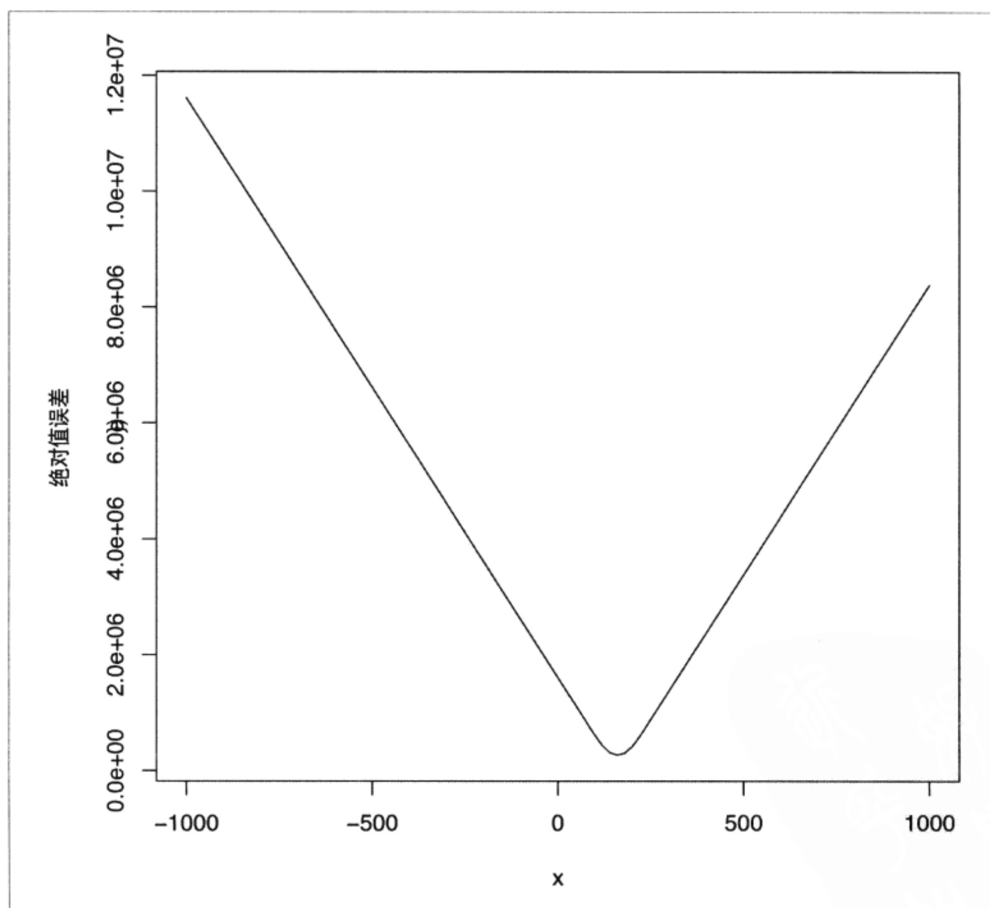


图7-5：改变a时的绝对值误差

因为有些强大的机器学习算法并不能与某些类型的误差函数相匹配，所以机器学习的精

妙之处就在于：你需要知道什么时候可以使用类似optim这样的简单工具，什么时候又需要使用更强大的工具。确实有算法可以解决绝对值误差优化，不过，这已不在本书讨论范围内。如果你真的对这个问题感兴趣，那么请你身边的高等数学高手给你讲讲凸优化（convex optimization）吧。

密码破译优化问题

和回归问题一样，几乎所有的机器学习算法，都可以看做最小化某种预测误差的优化问题。不过有时候，输入并不是简单的数值，因此使用optim函数计算一个单独点的误差函数并不能提供它周围点的足够信息。对于这样的问题，你可以简单地使用网格搜索，当然还有其他更好的方法。我们会关注其中一个很直观又很强大的方法。我们称这个方法的大致思想为随机优化：在可选的输入参数范围内，一定程度上随机地移动参数，不过要保证移动参数的方向是误差减少的方向而不是增加的方向。这个方法与好多你可能有所耳闻的优化算法有关，包括模拟退火算法、遗传算法以及马尔可夫链蒙特卡洛（Markov Chain Monte Carlo，MCMC）方法。本章中我们打算使用的算法是Metropolis方法，各种不同版本的Metropolis方法是众多流行的现代机器学习算法的基础。

我们将会通过本章的案例研究：密码破译，来演示Metropolis方法。由于我们定义的算法并不是一个非常高效的解码系统，因此它不能被真正地用在实际产品系统中，不过把它当做如何使用Metropolis方法的示例，却是一个不错的选择。更重要的是，这个例子将告诉我们：有些问题根本无法通过大多数类似optim函数这样现成的优化算法解决。

那么，让我们先来描述一下问题：已知一串使用替代密码（substitution cipher，又称替换密码或置换密码）加密的密文，我们该使用什么样的代替规则来解密密文得到原文呢？如果你不太熟悉替代密码也没关系，它是最简单的加密算法，它简单地把明文中的一个字母用另一个固定的字母替代，生成密文。ROT13[注2]加密可能是最有名的例子，当然你也可能听过凯撒密码（caesar cipher）。凯撒密码是一种非常简单的加密方式，你把每一个字母替换为字母表中它下一位的字母即可："a"变成"b"，"b"变成"c"，"c"变成"d"。（这是一个首尾相连的环："z"变成"a"）。

为了解释如何在R语言中使用加密算法，让我们先在R语言中实现凯撒密码：

```
english.letters <- c('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
                     'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
                     'w', 'x', 'y', 'z')

caesar.cipher <- list()
```

注2：　ROT13把每个字母替换成它在字母表后第13位的那个字母。"a"变成n，"b"变成"o"，以此类推。

```
inverse.caesar.cipher <- list()

for (index in 1:length(english.letters))
{
  caesar.cipher[[english.letters[index]]] <- english.letters[index %% 26 + 1]
  inverse.caesar.cipher[[english.letters[index %% 26 + 1]]] <- english.letters[index]
}

print(caesar.cipher)
```

有了加密算法之后，让我们写一个将明文加密成密文的函数：

```
apply.cipher.to.string <- function(string, cipher)
{
  output <- ''

  for (i in 1:nchar(string))
  {
    output <- paste(output, cipher[[substr(string, i, i)]], sep = '')
  }

  return(output)
}

apply.cipher.to.text <- function(text, cipher)
{
  output <- c()
  for (string in text)
  {
    output <- c(output, apply.cipher.to.string(string, cipher))
  }

  return(output)
}

apply.cipher.to.text(c('sample', 'text'), caesar.cipher)
```

现在我们已经有了一些处理密码的基本工具了，可以开始考虑如何破译密码了。就像处理线性回归的思路一样，我们会把破解替换密码的问题拆分成几个步骤：

1. 为每一种解密规则定义一个解密效果度量。
2. 定义一个基于目前已知的解密效果最优的解密规则的算法，对它进行随机修改来生成一个新的解密规则。
3. 定义一个算法，可以递进地生成破译效果逐渐变好的解密规则。

如何评估一个解密规则的质量效果呢？假设有人给你一串文本，并且告诉你这是经过一种替换加密算法加密的。例如，如果凯撒大帝给了你一张写着“wfoj wjej wjdj”的纸条，经过凯撒加密算法解密，你会发现这就是那句名言的：“veni vidi vici”[译注2]。

译注2：我来。我见。我征服。

假设，你仅仅收到一段密文，并且可以确定这段密文对应的原文是标准的英语。你该如何来解码它？

我们解决这个问题的方法是这样的：如果一个解密规则能够将密文转换成标准的英语，那就是一个好的解密规则。输入一个解密规则，你就能够在密文上运行这个解密规则，然后观察解密出来的输出是不是正常的英文。例如，假设我们有两个备选的解密规则：A和B，它们对应的结果分别如下：

- decrypt(T, A) = xgpk xkfk xkek
- decrypt(T, B) = veni vidi vici

看到这两个解密规则的输出结果，你会认为解密规则B显然要比解密规则A好。但是，为什么我们会做出这种直观决定呢？这是因为，我们发现解密规则B的输出更像是真正的英语，而解密规则A给出的结果则完全不知所云。我们可以通过一个程序来实现上面直观的思想，即利用一个词典数据库来帮助我们计算任何一串字符串是一个真正英文单词的概率。由一串高概率单词组成的文本串更可能是真正的英文，而由一串低概率单词组成的文本串更像是解密失败的产物。唯一的困难在于我们如何处理哪些不存在的单词。因为它们的概率是0，而程序在计算一段文本的概率时又需要将它所有单词的概率连乘起来，所以必须用一个非常小的数来替代0，比如，机器所能表示的最小浮点数，可以称为epsilon。一旦我们处理了这个边界情况，就可以使用词典数据库给两个解密串的解密质量进行打分：首先找到两个解密串中每个单词的概率，再把它们连乘起来得到整个解密串的概率，根据整个解密串的概率的高低来衡量两种解密规则的优劣。

通过词典数据库计算得到的解密串的概率，就是我们评估解密规则的误差函数。现在我们有了误差函数，我们的密码破译问题就完全变成一个优化问题了，只需要找到那个概率最高的解密串对应的解密规则就行了。

遗憾的是，找到概率最高的解密规则这样的问题并不是`optim`函数所能解决的。解密规则并不能图形化，也没有连续性。而没有连续性，`optim`函数就不知道它该往哪个方向去找更好的解密规则。因此，为了解决解密问题，我们需要一个全新的算法——也就是在本章前面提过的Metropolis方法。Metropolis算法的确适用于我们的问题，不过对于一般长度的加密串而言，它的运行速度非常慢[注3]。

Metropolis方法的基本思想是这样的：首先从一个随机的解密规则开始，然后通过多次循环优化它，直到最后它就有可能成为一个正确的解密规则。尽管这听上去有点像变魔术，但在实际应用它的表现不错，你可以自己做个试验来证实一下。而且，一旦我们拿

注3：　和其他语言例如Perl相比，R语言本身文本处理的速度就很慢，这无疑是雪上加霜。

到一个可能是正确的解密规则之后，就可以基于语义连贯性和语法来判断是否正确地解密了密文。

为了得到一条好的解密规则，首先从一条随机的解密规则开始，然后不断循环地改进解密规则，比如循环50 000次。因为每一次改进规则我们都倾向于朝着更好的方向改变，所以一遍一遍不断地重复这个过程，最终有可能得到一条效果还不错的解密规则，不过它不能保证循环50 000次就能得到我们想要的解密规则，也许需要50 000 000次。这就是为什么这个算法并不能应用于真实的密码破译系统：你不能保证算法在一定时间内给出一个结果，而在你忐忑不安地等待的时候，甚至都不知道算法是否在向正确的方向努力。这个案例研究是一个简单的示例，只是为了让你明白，如果有些复杂的问题使用某些算法无法解决的时候，可以尝试使用优化算法。

接下来具体阐述如何通过一个现有的解密规则来得到一个新的解密规则。我们通过一次只随机地改变当前规则中一处的方法来实现。也就是说，我们只改变解密规则中一个字母的对应关系。比如“a”当前对应的是“b”，我们修改当前的规则，把“a”对应到“q”。而因为替换密码的本质，这其实需要改变当前规则中的另外一个对应到“q”的字母规则，比如，也许当前规则下“c”对应的是“q”。为了保证我们得到的是一个合法的解密规则，我们必须把“c”对应到“b”。所以，我们通过现有解密规则来生成新解密规则的算法最终归结为交换两个字母的现有规则，其中一个字母是随机选的，而另一个字母是因为替换密码本身性质而决定的。

如果我们想简单一点，只有当新解密规则得到的解密串的概率变高的时候，才接受新的解密规则——称为贪心优化（greedy optimization）。遗憾的是，在这个例子中，贪心优化会让我们陷入坏的解密规则里面，因此通常使用下面的方法来决定使用原解密规则A还是新解密规则B：

1. 如果解密规则B解密出的解密串的概率大于解密规则A对应的解密串，那么我们用B代替A；
2. 如果解密规则B解密出的解密串的概率小于解密规则A对应的解密串，我们仍然有可能用B代替A，不过并不是每次都这么替换。具体地说，如果解密规则B对应的解密串的概率是`probability(T, B)`，而解密规则A对应的解密串的概率是`probability(T, A)`，则以`probability(T, B)/probability(T, A)`的概率从解密规则A替换到解密规则B。

注意： 如果这个比值看上去有点莫名其妙，没关系。这里真正重要的并不是这个具体的比值，而是“我们还是有一定概率来接受B”这个事实。这是我们不会陷入贪心优化的陷阱的关键原因。

在开始使用Metropolis方法处理不同的解密规则之前，我们需要前面提到过的处理解密规则的工具，实现它们的代码如下所示：

```
generate.random.cipher <- function()
{
  cipher <- list()

  inputs <- english.letters
  outputs <- english.letters[sample(1:length(english.letters),
  length(english.letters))]

  for (index in 1:length(english.letters))
  {
    cipher[[inputs[index]]] <- outputs[index]
  }

  return(cipher)
}

modify.cipher <- function(cipher, input, output)
{

  new.cipher <- cipher
  new.cipher[[input]] <- output
  old.output <- cipher[[input]]
  collateral.input <- names(which(sapply(names(cipher),
                      function (key) {cipher[[key]]}) == output))
  new.cipher[[collateral.input]] <- old.output
  return(new.cipher)
}

propose.modified.cipher <- function(cipher)
{
  input <- sample(names(cipher), 1)
  output <- sample(english.letters, 1)
  return(modify.cipher(cipher, input, output))
}
```

将这个生成新规则的工具和规则交换的工具结合使用，就可以弱化优化算法的贪心程度，从而避免浪费太多时间在明显不好的规则上，这些规则的概率比现有规则更小。为了以算法的方式来实现弱化版的贪心优化算法，我们计算`probability(T,B) / probability(T,A)`，再把它与一个0～1的随机数进行比较。如果随机数比`probability(T,B) /probability(T, A)`大，那么我们就更换当前的规则，否则就保持当前的规则不变。

为了计算一直不停提及的概率，我们已经创建了一个词典数据库，它包含了*/usr/share/dic/words*下每一个单词在维基百科上的出现次数，你可以通过下面的方式将它加载到R里面：

```
load('data/lexical_database.Rdata')
```

你可以查询几个简单的单词来看看它们在维基百科上出现的频率（见表7-2）：

```
lexical.database[['a']]
lexical.database[['the']]
lexical.database[['he']]
lexical.database[['she']]
lexical.database[['data']]
```

表7-2：词典数据库

单词	概率
a	0.01617576
the	0.05278924
he	0.003205034
she	0.0007412179
data	0.0002168354

准备好了词典数据库，我们需要一些计算文本概率的函数。首先，写一个从数据库读取概率的函数，这会让我们更容易处理那些概率为最小浮点数的“伪单词”。在R语言里最小的浮点数是.Machine$double.eps。

```
one.gram.probability <- function(one.gram, lexical.database = list())
{
  lexical.probability <- lexical.database[[one.gram]]

  if (is.null(lexical.probability) || is.na(lexical.probability))
  {
    return(.Machine$double.eps)
  }
  else
  {
    return(lexical.probability)
  }
}
```

现在我们有了一个可以计算单个单词概率的函数，还需要一个函数来计算一段文本的概率，首先将一串文本拆分成一组单词，然后分别计算这些单词的概率，再把它们的概率连乘起来就得到了一串文本的概率。遗憾的是，使用原始概率连乘的结果非常不稳定，因为连乘的结果非常接近0，有可能超出计算机所能表示的精度范围。因此，实际上计算的是对文本串的概率取对数之后的值，也就是把每个单词的概率取对数再累积求和之后的结果。这个结果就稳定多了。

```
log.probability.of.text <- function(text, cipher, lexical.database = list())
{
```

```
  log.probability <- 0.0

  for (string in text)
 {
   decrypted.string <- apply.cipher.to.string(string, cipher)
   log.probability <- log.probability +
   log(one.gram.probability(decrypted.string, lexical.database))
 }

  return(log.probability)
}
```

现在，所需要的单个模块都准备好了，我们可以写一个Metropolis方法的单步方法了：

```
metropolis.step <- function(text, cipher, lexical.database = list())
{
  proposed.cipher <- propose.modified.cipher(cipher)

  lp1 <- log.probability.of.text(text, cipher, lexical.database)
  lp2 <- log.probability.of.text(text, proposed.cipher, lexical.database)

  if (lp2 > lp1)
  {
    return(proposed.cipher)
  }
  else
  {
    a <- exp(lp2 - lp1)
    x <- runif(1)
    if (x < a)
    {
      return(proposed.cipher)
    }
    else
    {
      return(cipher)
    }
  }
}
```

现在，整个优化算法的每一个步都准备好了，让我们把它们组合起来，来看看它是如何工作的吧。首先把明文保存在一个R语言的字符串向量里面：

```
decrypted.text <- c('here', 'is', 'some', 'sample', 'text')
```

然后对这一串明文进行凯撒加密：

```
encrypted.text <- apply.cipher.to.text(decrypted.text, caesar.cipher)
```

我们首先创建一个随机的解密规则，然后运行50 000次Metropolis方法，把结果保存在一

个叫做results的数据框里面。针对每一次计算过程，都把解密串的对数概率值、当前的解密结果以及当前解密是否正确的变量保存下来。

警告： 当然，在现实中，如果尝试破译一串密文，是不可能知道破译是否成功的，这仅仅是一个出于演示目的的示例。

```
set.seed(1)
cipher <- generate.random.cipher()

results <- data.frame()

number.of.iterations <- 50000

for (iteration in 1:number.of.iterations)
{
  log.probability <- log.probability.of.text(encrypted.text, cipher, lexical.database)
  current.decrypted.text <- paste(apply.cipher.to.text(encrypted.text, cipher),
                                  collapse = ' ')
  correct.text <- as.numeric(current.decrypted.text == paste(decrypted.text,
                                                             collapse = ' '))
  results <- rbind(results,
                   data.frame(Iteration = iteration,
                              LogProbability = log.probability,
                              CurrentDecryptedText = current.decrypted.text,
                              CorrectText = correct.text))
  cipher <- metropolis.step(encrypted.text, cipher, lexical.database)
}

write.table(results, file = 'data/results.csv', row.names = FALSE, sep = '\t')
```

由于执行这一段代码需要花不少的时间，因此，在你等待程序运行的时候，可以来看一下表7-3中展示的结果的一部分内容：

表7-3：Metropolis方法的执行过程

循环次数	对数概率	当前的解密结果
1	-180.218266945586	lsps bk kfvs kjvhys zsrz
5000	-67.6077693543898	gene is same sfmpwe text
10000	-67.6077693543898	gene is same spmzoe text
15000	-66.7799669880591	gene is some scmhbe text
20000	-70.8114316132189	dene as some scmire text
25000	-39.8590155606438	gene as some simple text
30000	-39.8590155606438	gene as some simple text
35000	-39.8590155606438	gene as some simple text
40000	-35.784429416419	were as some simple text
45000	-37.0128944882928	were is some sample text
50000	-35.784429416419	were as some simple text

正如你看到的那样，我们在45 000次循环以后就已经非常接近真正的明文了，不过在循环结束的时候我们并没有找到正确的明文。如果仔细看一下结果，你就会发现，其实在第45 609次循环的时候，我们得到了正确的解密规则了，不过我们接下来把它换掉了。这正是这个目标函数的问题：它度量的是每一个单词是否是真正的英文单词，而不管它们连接起来是否符合英文语法或是否通顺。如果改变解密规则，能让你得到概率更高的单词，那么它就会改变解密规则，即使这会使解密串犯语法上的错误甚至完全不知所云。为了解决这个问题，你需要利用更多的英语知识，比如说连续两个单词出现在一起的概率。目前，它强调了使用这种特殊的目标函数所带来的优化问题的复杂性：有时候，优化算法得到的最优结果并不真正是你想要的结果。而且，当你使用优化算法来解决问题的时候，并不能完全离开人工监督。

我们的目标函数并没有足够的英语知识，这是一个问题，但是，事实上还有比这更复杂的问题。首先，Metropolis方法是一个随机优化算法，幸运的是，我们选的随机种子是1，如果我们选了一个不好的随机种子，可能需要执行数万亿次循环才能得到正确的解密规则。为了证实这一点，可以尝试选择不同的随机种子，再对每个种子循环1000次并分析其结果。

其次，Metropolis方法总是乐于放弃好的解密规则，正因为如此，它才是一个非贪心的算法，然而也正因为如此，如果观察足够长的时间，你就会发现它放弃了你真正想要的解密规则！在图7-6中，我们把每次循环后的对数概率值都画了出来，从图中你可以看到，这曲线是多么的崎岖。

有很多流行的方法来解决这种随机扰动问题。其中一个方法是随着循环次数越来越多，它越来越不可能接受没有变好的规则——也就是说它变得越来越贪心。这叫做模拟退火方法，这个方法很有用，你也可以尝试通过改变接受新解密规则的函数中那部分代码来尝试一下[注4]。

另一种方法接受这种随机性，它给出的优化结果并不是一个唯一结果，而是一个结果的概率分布。在我们例子中，这个方法行不通，不过对于那些优化的结果是一个数值的问题，能够生成多个可能结果是很有用的。

结束本章之前，我们希望你已经大致了解了优化算法是怎么实现的，并且你已经可以用它来解决一些复杂的机器学习问题了。前面提到的一些概念，在接下来的章节中，我们还会提及，特别是当讨论到推荐系统的时候。

注4：　事实上，通过设置一个参数，可以让optim函数使用模拟退火算法来代替标准算法。不过，在我们的例子中，并不能使用模拟退火版本的optim函数，因为它只能处理数值参数，而不能处理用来表示解密规则的这种数据结构。

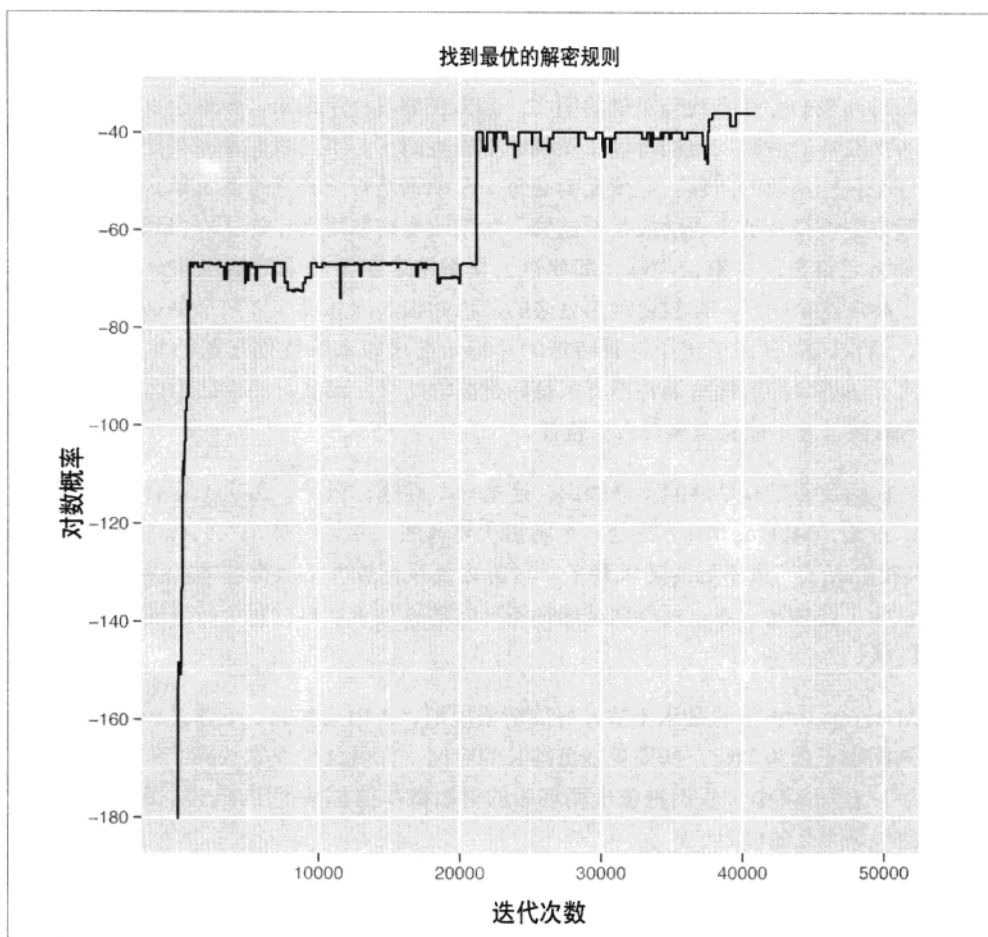


图7-6：Metropolis方法的执行过程：找寻最优的解密规则

第8章

PCA：构建股票市场指数

无监督学习

到目前为止，我们围绕数据所做的一切都是基于预测的任务：我们尝试对电子邮件或者网页访问量进行分类，在这些任务中，我们拥有训练用的样本数据集，其中的数据都已知正确的答案。正如本书曾提到的那样，当我们拥有已知正确答案的训练样本数据时，从中进行学习的过程称为有监督学习：发掘数据中的结构，并使用一个信号量进行衡量，在探索真实的模式这项工作中，我们是否做得很好。

但是，在没有任何已知答案指导的情况下，我们也经常想要发掘数据中的结构，这称为无监督学习。例如，我们也许想要进行数据降维，即把一个有很多列的表压缩成一个只有较少列的表。如果你需要处理很多的列，那么采取降维方法将会使得数据集更加容易理解。当你使用一列来代替多个列的时候，尽管明显损失了信息，但是在数据的可理解性上，通常获得了有价值的回报，在分析一个新的数据集时更是如此。

处理股市数据时，降维的优势就很大。例如，表8-1展示了真实的历史数据，这份数据是关于2010年01月02日~2011年05月25日期间25只股票的价格。

表8-1：历史股票价格

日期	ADC	AFL	…	UTR
2002-01-02	17.7	23.78	…	39.34
2002-01-03	16.14	23.52	…	39.49
…	…	…	…	…
2011-05-25	22.76	49.3	…	29.4

我们只列出了3列，实际上有25列，列太多了，不是很好处理。我们想要把25列的信息综合起来，得到一列，这一列可以告诉我们每天股市行情的好坏，这一列称为股市指数（index of the market）。那么，我们怎么才能构造一个股市指数呢？

主成分分析

针对上面的例子，最简单的方法就是称为主成分分析（Principal Components Analysis，PCA）的方法。PCA的主要思路是：创建一个25列的新数据集，根据每一列包含原始数据信息的多少，对新数据集中的列排序。排在第一的新列称为第一主成分（简称为主成分），它总是包含整个数据集的绝大部分结构。当我们数据集中的每一列都是强相关的时候，PCA特别有效。在这种情况下，你可以把原来相关的列替换为单独一列，这一列完全能反映原来列之间的潜在相关模式。

接下来让我们观察数据中的列之间有多大的相关性，从而看看PCA是否适用于这里。为此，首先需要把原始数据加载进R语言：

```
prices <- read.csv('data/stock_prices.csv')

prices[1,]
#        Date Stock Close
#1 2011-05-25   DTE 51.12
```

原始数据集并不是我们喜欢使用的格式，因此需要进行预处理。第一步，把数据集中的时间戳转换为正确编码的日期变量。这要用到lubridate程序包，这个程序包可以从CRAN获得。这个程序包中的ymd函数非常棒，它能把“年-月-日”这种格式的字符串转换成日期对象。

```
library('lubridate')

prices <- transform(prices, Date = ymd(Date))
```

一旦完成了这一步，我们就能使用reshape函数库中的cast函数，来创建一个和本章前面看到的表格相似的数据矩阵。在这个表中，每一行代表每一天，而每一列代表了不同的股票，接下来像下面这样进行操作：

```
library('reshape')

date.stock.matrix <- cast(prices, Date ~ Stock, value = 'Close')
```

在使用cast函数时，在波浪符号左边指定用数据源中哪些列作为输出矩阵的行，在波浪符号右边指定哪些列作为输出矩阵中的列。用value来指明输出矩阵中每一个元素的取值。

分析了这个生成结果——巨大的日期-股票矩阵之后，我们注意到缺失了一些元素。因此回到最初的prices数据集，删除那些缺失元素的数据，然后再运行cast函数：

```
prices <- subset(prices, Date != ymd('2002-02-01'))
prices <- subset(prices, Stock != 'DDR')

date.stock.matrix <- cast(prices, Date ~ Stock, value = 'Close')
```

删除了缺失元素的数据之后，我们重新生成想要的矩阵。做完这步之后，可以使用cor函数来找到这个矩阵中所有数字列之间的相关性。然后把相关性矩阵转换成一个数值向量，并且画一个相关性密度图，以此来获得两个直观认识：a）相关性的均值；b）低相关性出现的频率。

```
cor.matrix <- cor(date.stock.matrix[,2:ncol(date.stock.matrix)])

correlations <- as.numeric(cor.matrix)

ggplot(data.frame(Correlation = correlations),
            aes(x = Correlation, fill = 1)) +
  geom_density() +
  opts(legend.position = 'none')
```

我们画出的密度图如图8-1所示。正如我们可以看到的那样，大部分相关性是正数，因此PCA适用于这份数据集。

我们已经确信了可以使用PCA，怎么样才能在R语言中使用PCA呢？这又是一个R语言特别擅长的地方：整个PCA可以通过一行代码来完成。我们使用princomp函数来运行PCA：

```
pca <- princomp(date.stock.matrix[,2:ncol(date.stock.matrix)])
```

如果只在R语言命令行中输入pca，我们将看到一个关于主成分的简单汇总信息：

```
Call:
princomp(x = date.stock.matrix[, 2:ncol(date.stock.matrix)])

Standard deviations:
    Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6     Comp.7
29.1001249 20.4403404 12.6726924 11.4636450 8.4963820 8.1969345 5.5438308
    Comp.8     Comp.9    Comp.10    Comp.11    Comp.12    Comp.13    Comp.14
 5.1300931  4.7786752  4.2575099  3.3050931 2.6197715 2.4986181 2.1746125
   Comp.15    Comp.16    Comp.17    Comp.18    Comp.19    Comp.20    Comp.21
 1.9469475  1.8706240  1.6984043  1.6344116 1.2327471 1.1280913 0.9877634
   Comp.22    Comp.23    Comp.24
 0.8583681  0.7390626  0.4347983

24 variables and 2366 observations.
```

在这个汇总中，标准差（standard deviation）告诉我们每一个主成分的贡献率（成分方

差占总方差的比例）是多少。第一主成分称为Comp.1，贡献率是29%，而第二主成分的贡献率是20%。在末尾处，最后一个成分Comp.24带来的贡献率小于1%。这意味着仅仅采用第一主成分就能很好地对数据进行学习。

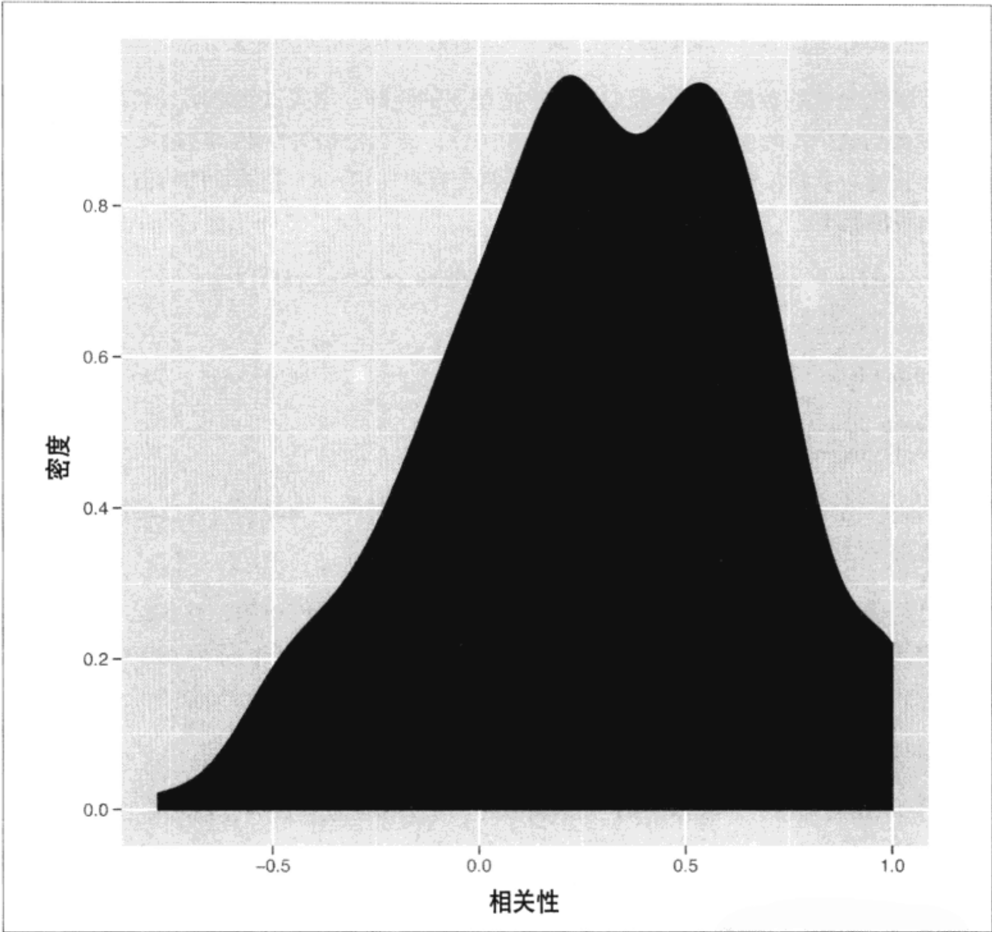


图8-1：股票价格数据中所有数值列之间的相关性

我们通过观察第一主成分的载荷量（loading）来更加细致地研究它。载荷值告诉我们，它给每一个主成分多大的权重。我们通过提取pca中princomp对象的loadings元素来获得这些信息。提取loadings后，可以获得一个大矩阵，它告诉我们源数据的25列中每一列有多少信息包含在每个主成分中。我们只对第一主成分感兴趣，所以只把pca载荷的第一列提取出来：

```
principal.component <- pca$loadings[,1]
```

完成这些之后，我们可以分析载荷的密度图，直观地了解第一主成分是如何形成的。

```
loadings <- as.numeric(principal.component)

ggplot(data.frame(Loading = loadings),
            aes(x = Loading, fill = 1)) +
  geom_density() +
  opts(legend.position = 'none')
```

结果如图8-2所示。这个结果有点让人疑惑，因为载荷有一个相当不错的分布，但是几乎全都是负数。一会儿，我们将看到这会导致什么问题。它实际上是个很小的麻烦，我们用一行代码就能解决。

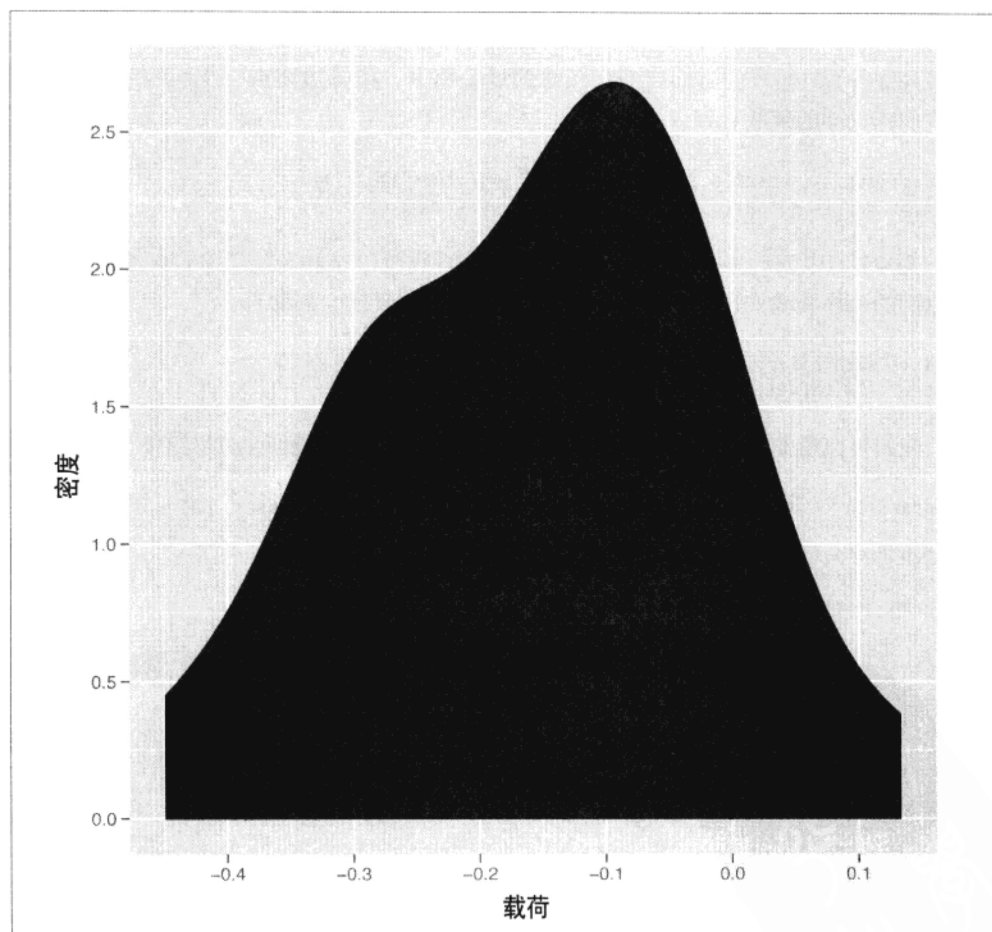


图8-2：主成分载荷

到目前为止我们已经获得了主成分，接下来可以把数据总结成一列了。可以使用predict函数完成这个目标：

```
market.index <- predict(pca)[,1]
```

如何才能知道这些预测值的效果呢？幸运的是，对这个实例我们可以很容易地判断结果好坏，因为可以把结果和著名的市场指数作比较。在本章中，我们使用道琼斯指数（Dow Jones Index），这里用DJI来代表它。

像下面这样把DJI加载进R：

```
dji.prices <- read.csv('data/DJI.csv')
dji.prices <- transform(dji.prices, Date = ymd(Date))
```

因为使用整个DJI运行的时间比我们预想的要长很多，所以需要取一个它的子集，仅仅获得我们感兴趣的那些日期。

```
dji.prices <- subset(dji.prices, Date > ymd('2001-12-31'))
dji.prices <- subset(dji.prices, Date != ymd('2002-02-01'))
```

然后，提取DJI中我们感兴趣的部分，也就是每日收盘价格和我们记录过的那些日期。因为它们的顺序和我们现在的数据集相反，用rev函数反转它们即可：

```
dji <- with(dji.prices, rev(Close))
dates <- with(dji.prices, rev(Date))
```

现在，我们可以绘制一些简单的图，将使用PCA生成的市场指数和DJI相比较：

```
comparison <- data.frame(Date = dates, MarketIndex = market.index, DJI = dji)

ggplot(comparison, aes(x = MarketIndex, y = DJI)) +
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE)
```

从图8-3可以看出，那些之前看上去烦人的负载荷，真的成了麻烦的源头：我们的指数和DJI负相关。

但是，我们可以很容易地解决这个麻烦。只需要对指数乘以-1，即可生成一个和DJI正相关的指数。

```
comparison <- transform(comparison, MarketIndex = -1 * MarketIndex)
```

现在可以再一次尝试进行比较了：

```
ggplot(comparison, aes(x = MarketIndex, y = DJI)) +
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE)
```

如图8-4所示，我们已经修正了指数的方向，并且它看上去和DJI真的很匹配。剩下的最后一件事情，就是获得我们的指数随着时间推移与DJI的趋势保持一致的程度。

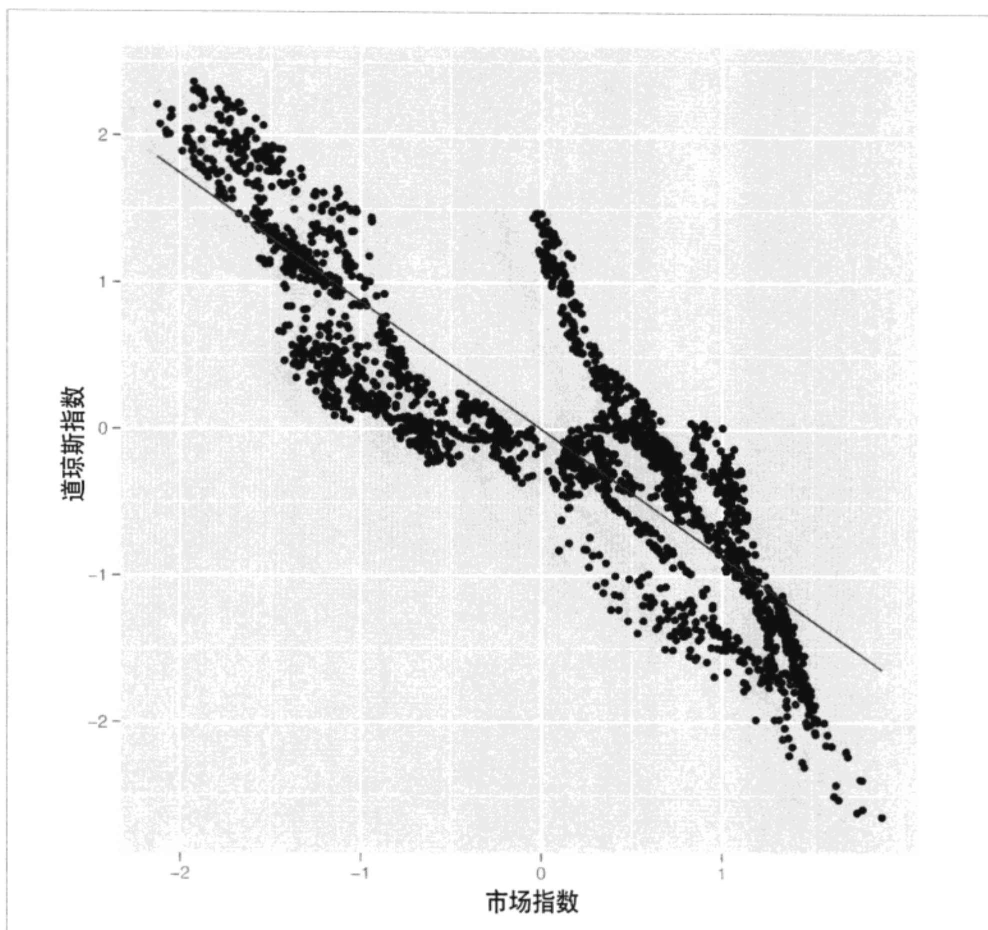


图8-3：PCA指数与道琼斯指数的初步比较

可以很容易进行比较。首先，使用melt函数获得一个数据框（data frame），它可以很容易地一次性对两个指数进行可视化。然后，我们对每个指数画出一条以日期为x轴，以价格为y轴的线。

```
alt.comparison <- melt(comparison, id.vars = 'Date')
names(alt.comparison) <- c('Date', 'Index', 'Price')

ggplot(alt.comparison, aes(x = Date, y = Price, group = Index, color = Index)) +
  geom_point() +
  geom_line()
```

这一次结果并不是很好，因为DJI都是很高的值，而我们的指数都是很小的值，但是可以使用scale函数解决这个问题，它把两个指数放到同一刻度下。

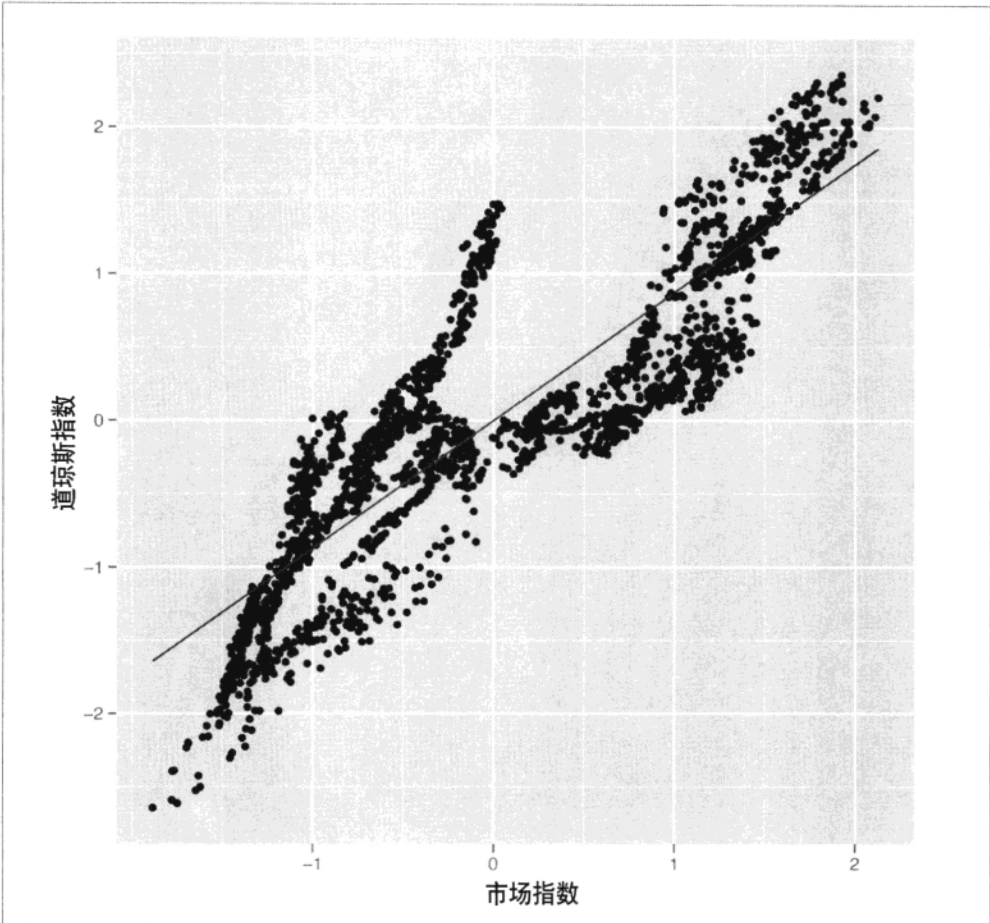


图8-4：改变刻度后的 PCA指数与道琼斯指数的比较

```
comparison <- transform(comparisonMarketIndex = -scale(MarketIndex))
comparison <- transform(comparisonDJI = scale(DJI))

alt.comparison <- melt(comparison, id.vars = 'Date')
names(alt.comparison) <- c('Date', 'Index', 'Price')

p <- ggplot(alt.comparison, aes(x = Date, y = Price, group = Index, color = Index)) +
  geom_point() +
  geom_line()

print(p)
```

然后重新画出曲线，并且检查结果，如图8-5所示。在图8-5中，我们完全使用PCA，并

且没有使用任何股市专业知识创建的市场指数，看上去与DJI的趋势保持得相当好。总之，原以为需要对股市状况的表示方法有深入思考才能得到这样的图，然而用PCA真的能够产生一幅股票价格的趋势图。我们觉得这相当令人惊叹。

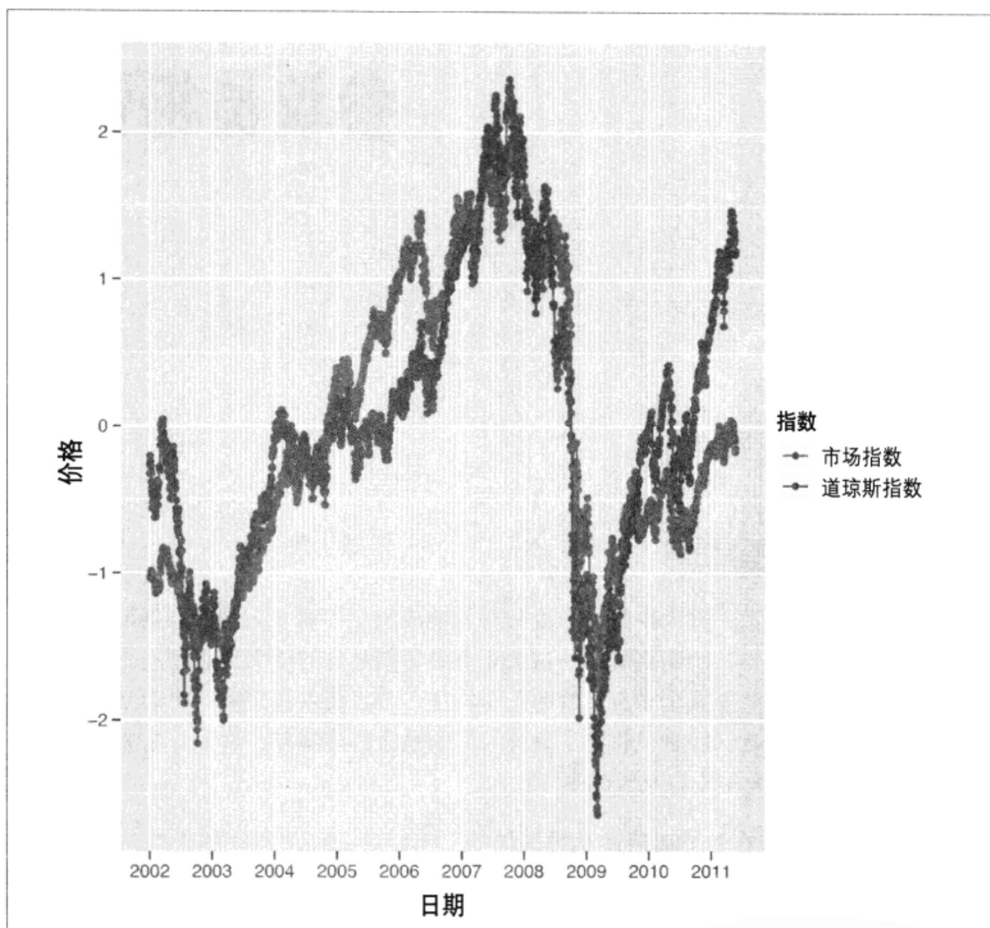


图 8-5：PCA指数与道琼斯指数随时间变化的比较

从这个例子可以看出，PCA是一个强大的工具，可以用于简化数据，甚至当你没有尝试预测任何事情时，就在发掘数据中的结构上事半功倍了。如果你对这个主题感兴趣，建议学习独立成分分析（Independent Component Analysis，ICA），它是PCA的一个变形，在某些PCA不能使用的情况下，可以很好地发挥作用。

第9章

MDS：可视化地研究参议员相似性

基于相似性聚类

很多时候，我们想知道一群人中的一个成员与其他成员之间有多么相似。例如，假设我们是一家品牌营销公司，刚刚完成了一份关于有潜力新品牌的研究调查问卷。在这份调查问卷中，我们向一群人展示了新品牌的几个特征，并且要求他们对这个新品牌的每个特征按五分制打分。我们同时也收集了目标人群的社会经济特征，例如：年龄、性别、种族、他们住址的邮政编码以及大概的年收入。

通过这份调查问卷，我们想搞清楚品牌如何吸引不同社会经济特征的人群。最重要的是，我们想要知道这个品牌是否有很大的吸引力。换个角度想这个问题，我们想看看那些最喜欢这个品牌特征的人们，是否有多种多样的社会经济特征。实现这一点的一种方法就是对问卷受访者的聚类结果进行可视化。然后就可以使用各种各样的视觉线索，识别出不同社会经济特征的成员。也就是说，我们想要看到大量不同性别、不同种族以及不同收入的人聚类到一起。

同样，我们想要使用这些知识来看看，相近的人群是如何基于品牌吸引力而聚类到一起的。我们也想看看一个聚类中有多少人，或者它与其他聚类的距离有多远。这也能告诉我们这个品牌的什么特征吸引了不同社会经济特征的人群。在提出这些问题时，我们使用这样的名词，如“近”和“远”，它们本身都是说明距离概念的词。因此，为了使聚类结果之间的距离更形象化，我们需要引入一些个体聚集的空间概念。

本章主旨是：对不同的观测记录，如何理解用距离的概念来阐明它们之间的相似性和相异性。这需要针对分析数据定义一些不同类型的距离矩阵。例如，在假设的品牌市场情形中，我们可利用调查规模的顺序属性以一种非常直接的方式发现不同受访者之间的距离：简单计算绝对差异。

可是，仅仅计算这些距离还不够。在本章中，我们将要介绍一种称为多维定标（multidimensional scaling，MDS）的技术，该技术的目的是基于观察值之间的距离度量进行聚类。通过MDS，我们可以只使用所有点之间的一个距离度量，就能将数据进行可视化。我们首先用一个用户对产品评分的演示实例来介绍MDS的基本原理，这个例子采用的是模拟数据。然后再使用真实的数据，这份真实数据是关于美国参议员记名投票的，我们以此展示：如何基于参议院成员的投票对他们进行聚类分析。

距离度量与多维定标简介

在开始正式介绍之前，假设已有一份非常简单的数据，这份数据中有4个用户，以及他们对6个产品的评分。每个用户按照要求对每个产品给出一个“拇指向上”或者“拇指向下”的评价，如果他们对某个产品没有任何意见，也可以忽略这个产品。有很多评价系统采用这种方式，包括Pandora和YouTube的评价系统。我们想要使用这个评分数据度量每个用户和其他用户之间有多大的相似性。

在这个简单的例子中，我们将会建立一个4 × 6的矩阵，在这个矩阵中行表示用户，列表示产品。我们将使用模拟评分填满这个矩阵，模拟评分随机地在“拇指向上”（1）、“拇指向下”（−1）以及忽略（0）中间选取一个作为每个“用户/产品对”的评分。为了完成这个任务，我们将要使用`sample`函数，这个函数随机地从一个向量`c(1, 0, -1)`中取值6次，采用放回式取值[译注1]。因为要用R的随机数产生器模拟数据，所以有一件事很重要，那就是你要设置和我们一样的随机数种子，这样才能让你的程序和我们的例子输出一样的结果。我们调用`set.seed()`函数设置种子，这里设置为851982。

```
set.seed(851982)
ex.matrix <- matrix(sample(c(-1,0,1), 24, replace=TRUE), nrow=4, ncol=6)
row.names(ex.matrix) <- c('A','B','C','D')
colnames(ex.matrix) <- c('P1','P2','P3','P4','P5','P6')
```

这段代码将建立一个4 × 6的矩阵，矩阵的每个元素对应用户（行）对产品（列）做出的评分。我们使用`row.names`和`colnames`这两个函数仅仅是为了使这个例子能描述得更加清晰：用户是A～D，而产品是1～6。当我们观察这个矩阵的时候，能够确切地看到示例数据集是什么样子的。例如，用户A给产品2和产品4“拇指向下”的评价，而且没有

译注1：放回式取值是指取过的值可以再次取。

评价其他任何产品。从另一行看，用户B给了产品1“拇指向下”的评价，但是他给产品3、4和5的评价是“拇指向上”。现在，我们需要使用这些差异来生成一个所有用户之间的距离度量。

```
ex.matrix
  P1 P2 P3 P4 P5 P6
A  0 -1  0 -1  0  0
B -1  0  1  1  1  0
C  0  0  0  1 -1  1
D  1  0  1 -1  0  0
```

使用这份数据生成一个距离度量的第一步是，摘要每个产品的用户评分，因为通过评分可以使用户和其他所有用户关联，也就是说，不像现在这样仅仅将用户和每个产品关联起来。思考这个问题的另一个方式就是，我们需要把这个N × M的矩阵，转化为一个N × N的矩阵，在新矩阵中，每个元素都代表了用户之间基于对产品评分所产生的关联。完成这一步的一个方法是，将现有的矩阵和它自己的转置相乘。这个乘法操作的效果是计算原来矩阵中每两行之间的相关性。

矩阵转置和矩阵乘法，应该是你在大学离散数学或者线性代数课程的前几周学过的知识。也许你曾经痛苦地手工进行那些转置计算，这得看你老师的性格了。幸运的是，现在R语言非常乐意帮你把转置工作完成了，这让你很开心吧?

注意： 本节介绍了矩阵操作以及使用它们构建评分数据距离度量的入门内容。如果你已经熟悉了，可以跳过这一节。

矩阵转置是指把一个矩阵反转，因此原来的行变成了列，原来的列成了行。从视觉上来看，转置把一个矩阵顺时针旋转90°，然后再把它垂直翻过来。例如，在前面的代码块中，我们可以看到“用户－评分”矩阵：ex.matrix，不过，在下面的代码块中，我们使用t函数转置它。现在，我们就有了一个“评分－用户”矩阵：

```
t(ex.matrix)
    A  B  C  D
P1  0 -1  0  1
P2 -1  0  0  0
P3  0  1  0  1
P4 -1  1  1 -1
P5  0  1 -1  0
P6  0  0  1  0
```

虽然矩阵的乘法更复杂一些，但是这里也只用到了基本算术运算。为了把两个矩阵相乘，我们循环遍历第一个矩阵的行和第二个矩阵的列。对于每一个“行－列”对，把每个行列元素对相乘，并且把相乘结果相加。在后面我们将看一个简单的例子，但是在做矩阵乘法时，要牢记一些重要的事情。首先，因为要把第一个矩阵的行元素和第二个矩

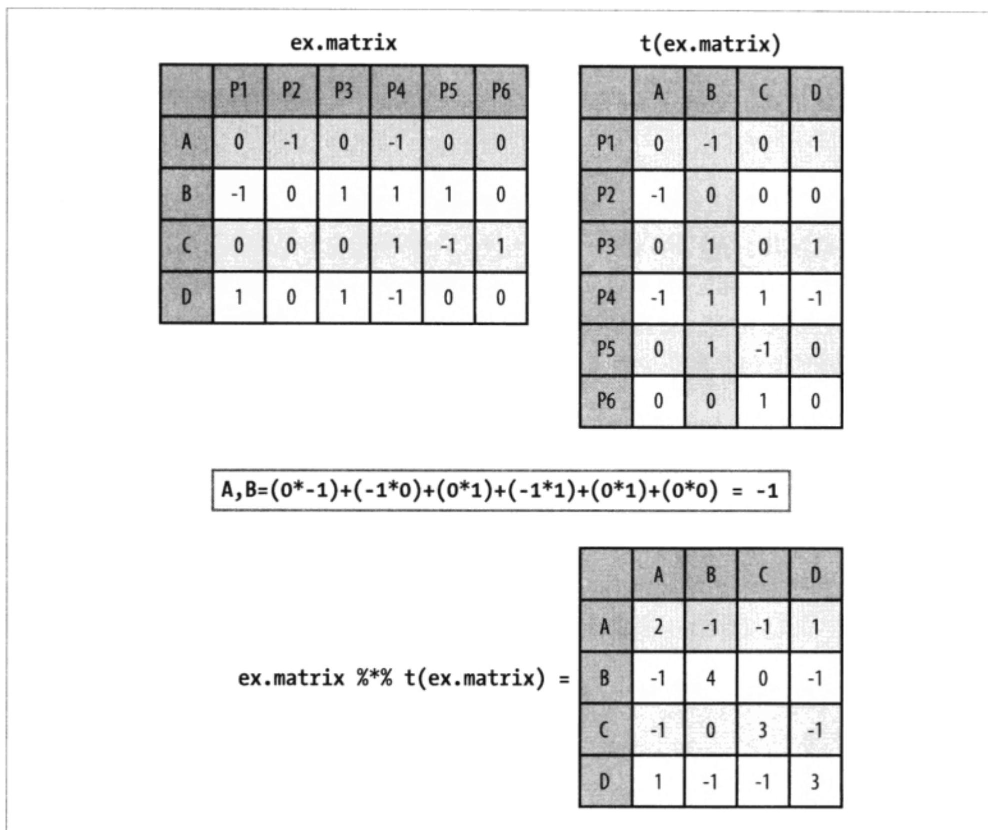
阵的列元素相乘，所以两个矩阵之间交叉的维度应该一致。在我们的例子中，不能简单地把矩阵ex.matrix与一个2×2矩阵相乘；如果你试试这么做，就会看到算术运算无法正常进行。据此推断，矩阵乘法的结果，将会总是和第一个矩阵有相同个数的行，和第二个矩阵有相同个数的列。

一个直接推论是：在矩阵乘法中，顺序是有关系的。在本例中，我们希望得到一个能够摘要用户之间差异的矩阵，所以将“用户－评分”矩阵和它自己的 “评分－用户”转置矩阵相乘，得到一个乘积。如果以完全相反的方式来做，例如，将“评分－用户”矩阵和它自己的转置矩阵相乘，将得到反映评分之间差异的矩阵。在其他情况下，这样做也许是我们感兴趣的，但是现在它没有什么用处。

在图9-1中，我们演示了如何进行矩阵乘法。左上角是ex.matrix，右上角是它的转置矩阵。有了这两个矩阵之后，我们把它们按照从左到右的顺序相乘。作为一个矩阵乘法的例子，我们突出显示了ex.matrix的行A及其转置矩阵的列B。在紧挨着这两个矩阵的下面，详细展示了矩阵乘法的算术运算。算术运算非常简单，取第一个矩阵一行中的元素，然后将其和第二个矩阵列中对应的元素相乘。然后把所有乘积相加。你也看到了，在“用户－用户”结果矩阵中元素[A，B]是-1，也就是“行－列”相乘后求和的结果。

```
ex.mult <- ex.matrix %*% t(ex.matrix)
ex.mult
   A  B  C  D
A  2 -1 -1  1
B -1  4  0 -1
C -1  0  3 -1
D  1 -1 -1  3
```

在图9-1中我们也介绍了一些R语言的符号，这些符号在前面生成矩阵乘积的代码块中反复出现。在R语言中，可以使用%*%操作符进行矩阵乘法。对新矩阵的解释非常直观。因为我们使用了1、－1和0的编码机制，所以非对角线元素的值反映了：在已知两个用户都进行了评分的那些产品的情况下，用户对产品的评分大体上是一致的（正值），还是不一致的（负值），就我们的例子来说，就是那些非零元素。非对角线的元素的正值越大，则两个用户的评分就越一致；同样，负值越小，则两个用户的评分就越不一致。因为我们最初的矩阵元素是随机的，所以不同用户之间只有很小的差异，于是没有非对角线元素的值是大于1或者小于－1的。对角线元素的值只是反映了每个用户对多少个产品进行了评分。



ex.matrix

	P1	P2	P3	P4	P5	P6
A	0	-1	0	-1	0	0
B	-1	0	1	1	1	0
C	0	0	0	1	-1	1
D	1	0	1	-1	0	0

t(ex.matrix)

	A	B	C	D
P1	0	-1	0	1
P2	-1	0	0	0
P3	0	1	0	1
P4	-1	1	1	-1
P5	0	1	-1	0
P6	0	0	1	0

A,B=(0*-1)+(-1*0)+(0*1)+(-1*1)+(0*1)+(0*0) = -1

ex.matrix %*% t(ex.matrix) =

	A	B	C	D
A	2	-1	-1	1
B	-1	4	0	-1
C	-1	0	3	-1
D	1	-1	-1	3

图9-1：矩阵乘法举例

现在，我们有了关于用户之间差异的摘要，这在一定程度上是有用的。例如，用户A和用户D都给了产品4反对票；可是，用户D喜欢产品1和产品3，而用户A根本没有对它们进行评价。因此，从它们都给出投票信息的那些产品的角度来看，我们可以说这两个用户是相似的，因此我们有一个值为1的元素对应他们之间的关系。很遗憾，这个结果传达的信息是有限的，因为我们只能对用户的共有评分记录给出一些解释。但是我们想要的方法是可以把用户评分记录的差异泛化到更丰富的表达程度。

为此，我们将在多维空间中引入欧氏距离（Euclidean distance）的概念。在一维空间、二维空间或者三维空间中，欧氏距离是我们直观上所感受到的距离的一个公式化描述。为了计算空间中两点之间的欧氏距离，我们测量它们之间最短的直线距离。在本例中，我们想要基于上面的矩阵乘积所定义的整体相似性和差异性度量，来计算所有用户之间的欧氏距离。

为了实现这一点，我们将把每个用户的所有评分作为一个向量。为了比较用户A和用户

B，可以把他们对应的向量相减，然后对差进行平方运算，将平方结果加到一起，最后对加和结果求平方根。这样就得到了用户A的所有评分和用户B的所有评分之间的欧氏距离。

我们可以使用R语言中的基本函数sum和sqrt来“手工”计算欧氏距离。在下面的代码块中，我们展示了对用户A和用户D所做的这种计算，结果大概是2.236。幸运的是，因为计算一个矩阵所有两行之间的距离是一种非常常见的操作，所以R语言有个基本函数叫做dist，这个函数正是用来做这些计算的，而它会返回一个关于距离的矩阵，我们把这个矩阵称作“距离矩阵”（distance matrix）。dist函数可以使用几种不同的距离度量方法产生一个距离矩阵，但是我们仍会坚持使用欧氏距离，这也是该函数的默认方法。

```
 sqrt(sum((ex.mult[1,]-ex.mult[4,])^2))
[1] 2.236068

 ex.dist <- dist(ex.mult)
 ex.dist

         A        B        C
B 6.244998
C 5.477226 5.000000
D 2.236068 6.782330 6.082763
```

现在，ex.dist变量中保存了距离矩阵。正如你从代码块中看到的那样，这个矩阵实际上只是整个距离矩阵的下三角部分。因为行X到行Y之间的距离与行Y到行X之间的距离是一样的，所以距离矩阵一定是对称矩阵，于是像这样只显示距离矩阵的下三角部分很常见。显示距离矩阵的上三角是冗余的，一般不会这么做。但是你可以在调用dist函数的时候，设置参数upper=TRUE覆盖只显示下三角矩阵这个默认值。

正如我们可以在ex.dist矩阵的下三角中的值所看到的，用户A和用户D是距离最近的，而用户D和用户B是距离最远的。现在，我们对基于用户对产品评分的用户之间相似性，有了更加清晰的感性认识，不过，如果我们可以获得关于这种差异的视觉认识，就更好了。这正是MDS可以发挥作用的地方，它可以基于我们刚刚计算出的距离生成一个用户空间布局。

MDS是一个统计技术集合，用于可视化地描述距离集合中的相似性和差异性。对于经典的MDS——我们将会在本章中用到，它的整个处理过程包括：输入一个包含数据集中任意两个数据点之间距离的距离矩阵，返回一个坐标集合，这个集合可以近似反映每对数据点之间的距离。之所以说是近似反映，是因为在二维空间中很可能不存在被一组距离分开的点集。例如，在二维空间中，不可能找到4个彼此间距离都是1的点。（请注意，3个彼此之间距离都是1的点，是一个等边三角形的顶点。因此，不可能有另外一个点到这个三角形的3个顶点的距离都是1。）

经典的MDS使用一个特别的距离矩阵近似方法，因此它也是另外一个用于机器学习的优化算法的例子。当然，经典MDS背后的近似算法，也可以用到三维空间或者4维空间，但是我们的目标是获得一个表达我们数据的方式，以便于方便地进行可视化。

注意： 对于本章中的所有实例，我们都将使用MDS在二维空间中刻画数据。这是最常见的MDS使用方式，因为这样使我们可以非常简单地在一个坐标图中将数据可视化。但是，毫无争议的是，MDS也可以在更高维空间使用。例如，三维可视化可能揭示出聚类结果中数据点在第三个维度中的不同层次。

经典的MDS例程是R语言基础函数`cmdscale`的一个组成部分，而且这个函数只要求输入一个距离矩阵，例如`ex.dist`。`cmdscale`这个函数将默认地在二维空间中计算MDS，但是你可以使用k这个参数改变这个设置。因为我们只对在二维空间中刻画距离数据感兴趣，所以在下面的代码块中使用了默认参数设置，并且使用R语言的基础图形把结果画出来。

```
ex.mds <- cmdscale(ex.dist)
plot(ex.mds, type='n')
text(ex.mds, c('A','B','C','D'))
```

从图9-2中我们可以看到，用户A和用户D确实在图的中间靠右位置被聚类到一起。可是，用户B和用户C根本就没有形成聚类。从我们的数据来看，确实可以看到用户A和用户D具有某种程度的相似口味，但是，我们需要获得更多数据以及（或者）用户，才能弄清楚为什么用户B和用户C被聚类到一起。需要注意的是，尽管可以看到用户A和用户D是如何聚类的，以及类似的用户B和用户C是如何没有被聚类的，但是关于如何解释这些距离，我们不能给出任何实质性的说法。也就是说，我们知道用户A和用户D更加相似，因为它们在坐标平面上更加接近，但是我们不能使用它们之间的量化距离来解释它们究竟有多么相似，或者解释用户B和用户C之间到底有多么不相似。MDS产生的具体距离数值是由MDS算法制造出来的，根据它，不能给出多少实质性的解释。

在下一节，我们将完成一个案例研究，它和刚刚使用的演示例子很相似，但是将使用来自美国参议院的记名投票的真实数据。这份数据比演示例子的数据集大多了，而我们将使用它来展示美国参议院成员在历届国会上分别是如何被聚类的。我们将使用参议院的记名投票记录来产生距离度量，然后将使用MDS在二维空间中将参议员聚类。

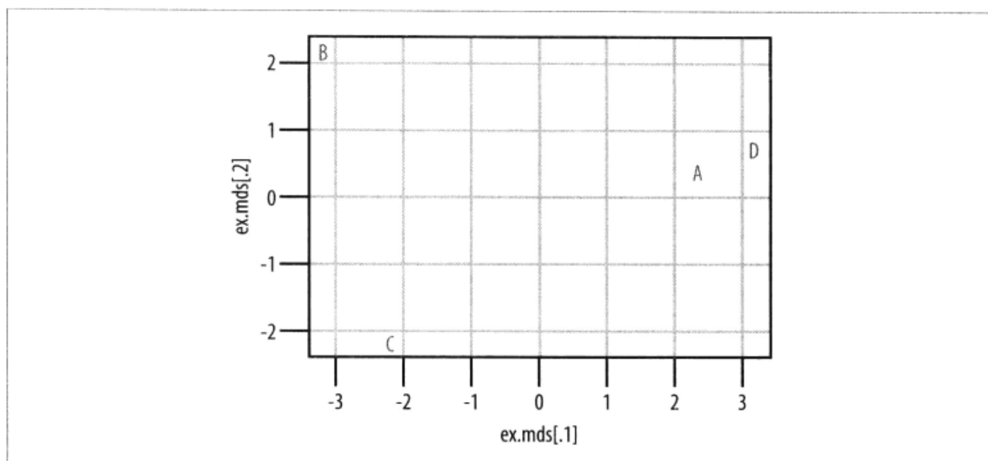


图9-2：关于用户对产品模拟评分数据的MDS图

如何对美国参议员做聚类

> 当前的国会——第111届国会是当代历史中意识形态两极化最严重的一届。在白宫和参议院中，最保守的民主党都要比最开明的共和党还要开明。如果把国会的“中心”定义为两党重叠的部分，那么这个中心已经消失了。
>
> ——William A. Galston，布鲁金斯学会（2010）

我们经常从William A. Galston那里听到与此类似的评论，他是在布鲁金斯学会（The Brookings Institute）从事政府管理研究的高级研究员，他声称美国国会的两极化处在历史最高点[WA10]。原因当然不言自明。流行出版物经常描述这种印象，而且美国的主流媒体也经常夸大这些差异。如果把立法过程陷入泥潭当做这种两极化的副作用，那么可以期望把立法结果当做两极分化程度的粗略度量。在第110届国会中，提出将近14 000份立法案，但是只有499项法案（3.3%）真正变成了法律[PS08]。事实上，在这449项法案中，第144项只是建议改变联邦大楼的名字。

但是，现在美国国会真的比以前更加两极分化了吗？尽管我们也许相信这是事实，但是更愿意提供证据。我们将在此使用的方法是，在不考虑党派区别的条件下，用MDS对参议员聚类进行可视化，以此观察两党成员是否存有混合在一起的情况。可是，在做这些事情之前，我们需要一个参议员之间距离的衡量标准。

幸运的是，我们可以使用立法者的公开记录创建一个合理的距离度量。我们在这里将使用立法者的投票记录。和前一部分的例子一样，我们可以使用记名投票记录来看一个立

法者是支持还是反对一项提案。就像前面例子中用户给出的“拇指向上”或者“拇指向下”的评价一样，立法者使用“是（赞成）”或者“否（反对）”对法案进行投票。

这里需要给不熟悉美国立法程序的读者解释一下，记名投票是美国国会任何一个会议室中最基本的议会程序之一。顾名思义，它是美国众议院和参议院在任何提案生效前都要进行的一项程序。两院分别通过不同的机制发起一个记名投票，但是结果基本上是一致的。记名投票记录了每个立法者对一项提案所做出的反应。正如之前提到过的，这种投票一般采用“赞成”或“反对”的形式，但是稍后我们将会看到实际的投票结果要更加复杂一些。

这份记名投票数据完全可用于衡量立法者之间的相似性和差异性，并且对于研究美国国会的政治学家来说，是无价的资源。这份数据是如此有价值，以至于两位政治学家创建了一个统一的可以下载的资源。Keith Poole（乔治亚大学）和Howard Rosenthal（纽约大学）维护了一个网站*http://www.voteview.com/*，它是一个存放所有美国记名投票数据的仓库，从第1届国会一直到写这本书时的最近一次国会：第111届国会。虽然在这个网站上也可以获得许多其他的数据集，但是我们将只使用美国参议院第101届到111届国会的记名投票数据。这些数据放在本书官网的本章数据文件夹中。

在本章剩余的部分中，我们将逐一介绍在这份记名投票数据上运行MDS算法的代码。一旦计算出距离的近似值，我们就可以通过可视化结果来回答这样的问题：在使用记名投票记录进行聚类时，来自不同党派的参议员会被聚类到一起吗？

分析参议员记名投票数据

正如之前提到的那样，我们已经收集了从第101届～111届国会的所有参议院记名投票数据，并且把它们放到本章的数据文件夹下。和之前做过的一样，我们将以加载这份数据作为这个案例研究的开始，并且对它做一些分析。在这个过程中，我们将使用两个R语言函数库：第一个是foreign库，稍后将详细讨论它；第二个是`ggplot2`库，将使用它来可视化MDS算法的结果。这些数据文件存放在*data/roll_call/*文件夹下，因此我们使用`list.files`函数创建一个叫做`data.files`的字符向量，它包含了所有数据文件的文件名。

```
library(foreign)
library(ggplot2)

data.dir <- "data/roll_call/"
data.files <- list.files(data.dir)
```

当我们查看`data.files`变量时，可能注意到这些数据文件的扩展名和本书前面其他章节用于案例研究的文本文件有所不同。扩展名`.dta`对应的是Stata数据文件。Stata是一个商业统计计算软件，它曾经在学术界非常流行，特别是政治学家喜欢用。因为Poole和

Rosenthal决定用这种格式公开这份数据，所以我们需要一个把这份数据加载到R语言的方法。

```
data.files
 [1] "sen101kh.dta"  "sen102kh.dta"
 [3] "sen103kh.dta"  "sen104kh.dta"
 [5] "sen105kh.dta"  "sen106kh.dta"
 [7] "sen107kh.dta"  "sen108kh_7.dta"
 [9] "sen109kh.dta"  "sen110kh_2008.dta"
[11] "sen111kh.dta"
```

深入了解一下foreign程序包，其设计目标是读取大量的外部数据文件到R语言的数据框中，外部数据文件包括：S、SAS、SPSS、Systat、dBase以及其他很多软件的数据文件。对于本次分析来说，我们要分析的数据来自11届国会，从第101届～第111届[注1]。我们将会把所有数据存放到一个对象里，这样就可以一次性处理所有数据。我们将会看到，这些数据集是相对较小的，因此在这个案例中不用关心内存的问题。为了整合数据集，我们将同时使用lapply函数和read.dta函数。

```
rollcall.data <- lapply(data.files,
function(f) read.dta(paste(data.dir, f, sep=""), convert.factors=FALSE))
```

现在，我们有了关于所有记名投票的11个数据框，它们分别存放在rollcall.data变量中。当我们查看第一个数据框（第101届国会的数据）的维度时，看到它有103行和647列。我们进一步查看这个数据框的头部，就会看到这些行和列里都有什么。在查看数据的头部时，有两个重要的事情需要注意。首先，每一行都对应了美国国会中的一位投票者。其次，数据框的前9列包含了那些投票者的身份信息，而剩余的列才是实际的投票。在我们可以更近一步之前，需要了解一下这些身份信息。

```
 dim(rollcall.data[[1]])
[1] 103 647

 head(rollcall.data[[1]])
  cong    id state dist  lstate party eh1 eh2       name  V1 V2 V3 ... V638
1  101 99908    99    0 USA       200   0   0 BUSH         1  1  1 ...    1
2  101 14659    41    0 ALABAMA   100   0   1 SHELBY, RIC  1  1  1 ...    6
3  101 14705    41    0 ALABAMA   100   0   1 HEFLIN, HOW  1  1  1 ...    6
4  101 12109    81    0 ALASKA    200   0   1 STEVENS, TH  1  1  1 ...    1
5  101 14907    81    0 ALASKA    200   0   1 MURKOWSKI,   1  1  1 ...    6
6  101 14502    61    0 ARIZONA   100   0   1 DECONCINI,   1  1  1 ...    6
```

某些列的意义是显而易见的，比如lstate和name，但是eh1和eh2说的是什么呢？值得庆幸的是，Poole和Rosenthal为所有记名投票数据提供了一个编码手册。这个编码手册关于

注1： Poole和Rosenthal也提供了一个叫做readKH的R语言函数，用于读取.ord数据类型。想要了解更多信息，请参考*http://rss.acs.unt.edu/Rdoc/library/pscl/html/readKH.html*。

第101届国会的部分可以在*http://www.voteview.com/senate101.htm*获得，并且在例9-1中也复制了一份。这个编码手册特别有用，因为它不仅解释了前9列中每一列都包含什么，而且说明了每个投票是怎么编码的，这也是我们马上需要留意的。

例9-1：Poole和Rosenthal提供的记名投票数据编码

```
1. Congress Number
2. ICPSR ID Number: 5 digit code assigned by the ICPSR as
      corrected by Howard Rosenthal and myself.
3. State Code: 2 digit ICPSR State Code.
4. Congressional District Number (0 if Senate)
5. State Name
6. Party Code: 100 = Dem., 200 = Repub. (See PARTY3.DAT)
7. Occupancy: ICPSR Occupancy Code -- 0=only occupant; 1=1st occupant; 2=2nd occupant; etc.
8. Last Means of Attaining Office: ICPSR Attain-Office Code -- 1=general election;
      2=special election; 3=elected by state legislature; 5=appointed
9. Name
10 - to the number of roll calls + 10: Roll Call Data --
             0=not a member, 1=Yea, 2=Paired Yea, 3=Announced Yea,
                   4=Announced Nay, 5=Paired Nay, 6=Nay,
             7=Present (some Congresses, also not used some Congresses),
             8=Present (some Congresses, also not used some Congresses),
             9=Not Voting
```

对我们来说，只关心投票者的名字以及他们的党派和实际的投票。因此，首先我们把记名投票数据组织成特定格式，可以用它创建一个关于投票的合理距离度量。正如我们在例9-1中所看到的，参议院的记名投票并不是简单的“赞成”或者“反对”，这两种投票还有Announced（公开的）和Paired（配对的）两种形式，同时还有Present（出席）投票，也就是说，一个参议员在某个特定法案的投票中已经弃权，但是他在投票的时候出席了国会。有时候，也有一些参议员没有出席投票，或者甚至还没被选进参议院。已经有了这么多种可能的投票方式，那么我们该怎样把它们利用起来，转换成便于使用的形式来度量参议员之间的距离呢？

我们采用的办法是，通过把相似的投票类型聚集到一起，来简化数据的编码方式。例如，“配对投票”的程序是这样的，国会成员知道自己无法出席某个特定记名投票，他们可以把自己的投票和另外一位将会和他投相反票的国会成员配对。对于“公开投票”而言，是议会制定的对参议院或者白宫某个正在进行的投票所做出的仲裁。可是对于我们来说，并不怎么关心产生这些投票的机制，关心的是这些投票展现出来的倾向性，如支持或者反对。因此，一种聚集方式就是，将所有类型的赞成票和反对票分别放进不同的组中。基于同样的逻辑，我们也可以把所有类型的无效投票放到同一组。

```
rollcall.simplified <- function(df) {
    no.pres <- subset(df, state < 99)
    for(i in 10:ncol(no.pres)) {
        no.pres[,i] <- ifelse(no.pres[,i] > 6, 0, no.pres[,i])
        no.pres[,i] <- ifelse(no.pres[,i] > 0 & no.pres[,i] < 4, 1, no.pres[,i])
```

```
        no.pres[,i] <- ifelse(no.pres[,i] > 1, -1, no.pres[,i])
    }
    return(as.matrix(no.pres[,10:ncol(no.pres)]))
}

rollcall.simple <- lapply(rollcall.data, rollcall.simplified)
```

图9-3演示了将要用于在数据分析中简化Poole和Rosenthal所使用编码的过程。和上一个模拟数据的例子一样，我们将所有的赞成票编码为+1，所有的反对票编码为-1，而没有投票的观察值编码为0。对于距离度量来说，这是一种非常直观的数据编码应用。现在，我们需要用这种编码方式给数据编码，而且也只需要从数据框中提取投票信息，以便在后面的步骤中进行矩阵操作。



图9-3：简化记名投票记录的方法

为了实现目标，我们定义了`rollcall.simplified`函数，这个函数的唯一参数就是记名投票数据，并且返回一个“参议员—投票”矩阵，这个矩阵使用的是简化后的编码。你将会注意到，这个函数的第一步把`State`那一列的值等于99的观察值都删除了。编码值为99的州对应的是美国的副总统，而副总统很少投票，所以把关于他的数据删除了。然后，使用`ifelse`命令对矩阵中剩下的所有列进行向量化数值比较。请注意，我们进行的比较是顺序相关的。首先把所有的无效投票（所有编码大于6的投票）编码设置为0；然后，寻找编码大于0并且小于4的投票，也就是赞成票，然后把它们的编码转换成1；最后，所有编码大于4的是反对票，把它们的编码设置为−1。

现在，我们把记名投票处理成了和前一部分模拟数据例子开始时相同的形式，并且可以使用与处理模拟评分数据完全相同的方式继续进行处理。在本章后面的内容中，我们将产生这份数据的MDS，并且以可视化的方式研究它。

研究通过国会对参议员进行MDS聚类

和前面一样，第一步是使用“参议员—投票”矩阵创建一个“参议员—参议员”距离矩阵，再对这个矩阵应用MDS算法。我们将使用lapply函数对每一届国会分别进行这种转换。首先进行矩阵乘法，并将结果存放在变量rollcall.dist中，然后，再通过lapply函数调用cmdscale函数实现MDS。关于MDS操作，有两点需要注意。第一，cmdscale函数默认在二维空间进行MDS计算，因此设置k=2是多此一举。可是，这样做还是有用的，在代码共享时，显式地指出所进行的操作更容易让人明白，因此这样做是最好的方式。第二，我们把所有点都与－1相乘。这样做完全是为了可视化，把所有点的x坐标都翻转了，我们会看到，民主党人被放到左边，共和党人被放到右边。在美国的政治背景下，这是一个有用的视觉暗示，因为我们一般认为民主党人的意识形态偏左，而共和党人的意识形态偏右。

注意： 你也许已经猜到，我们是在完成MDS可视化之后，才发现民主党人出现在x轴右边，共和党人出现在左边。做好数据分析最重要的就是，在完成计算之后，对如何提升方法的效果和结果的表达进行灵活处理和批判性思考。因此，尽管在这里我们直接给出这样的处理方式，但是你要知道，这是在经过第一次尝试之后，我们才决定在图像中翻转x轴的。

```
rollcall.dist <- lapply(rollcall.simple, function(m) dist(m %*% t(m)))

rollcall.mds <- lapply(rollcall.dist,
    function(d) as.data.frame((cmdscale(d, k=2)) * -1))
```

下一步，我们需要把相应的身份数据加到rollcall.mds的坐标点数据框中，以便在加入了党派数据后对其可视化。在下一个代码块中，我们将对rollcall.mds列表使用一个简单的for循环来完成这个工作。首先，把坐标点的列名分别设成x和y。然后，我们访问rollcall.data中的原始记名投票数据框，并且提取参议员姓名这一列。还记得吗？我们得首先去掉副总统的数据。此外，有些参议员的名字包含了“名”和“姓”，但是更多是只有“姓”。为了保持一致性，用逗号拆分字符串向量name（姓名），并将拆分得到的前半部分保存在变量congress.names中。最后，使用转换函数把党派信息作为因子添加进去，同时添加国会的编号。

```
congresses <- 101:111
for(i in 1:length(rollcall.mds)) {
    names(rollcall.mds[[i]]) <- c("x", "y")
    congress <- subset(rollcall.data[[i]], state < 99)
    congress.names <- sapply(as.character(congress$name),
```

```
        function(n) strsplit(n, "[, ]")[[1]][1])
    rollcall.mds[[i]] <- transform(rollcall.mds[[i]], name=congress.names,
        party=as.factor(congress$party), congress=congresses[i])
}

 head(rollcall.mds[[1]])
            x        y      name party congress
2  -11.44068 293.0001    SHELBY   100      101
3  283.82580 132.4369    HEFLIN   100      101
4  885.85564 430.3451   STEVENS   200      101
5 1714.21327 185.5262 MURKOWSKI   200      101
6 -843.58421 220.1038 DECONCINI   100      101
7 1594.50998 225.8166    MCCAIN   200      101
```

在添加了上下文数据之后，观察rollcall.mds中第一个国会信息的前几个元素，可以看见这个数据框部分信息。在本章包含的R语言代码中，有很多用于循环数据框列表的命令，其目的是为每一届国会分别创建一个可视化结果。为了方便起见，我们在这里仅仅包含这份代码的一部分。这里列出的代码只画出第110届国会的数据，但是只需对代码进行简单修改就可以画出其他任何一届国会的数据。

```
cong.110 <- rollcall.mds[[9]]
base.110 <- ggplot(cong.110, aes(x=x, y=y))+scale_size(to=c(2,2), legend=FALSE)+
    scale_alpha(legend=FALSE)+theme_bw()+
    opts(axis.ticks=theme_blank(), axis.text.x=theme_blank(),
        axis.text.y=theme_blank(),
        title="Roll Call Vote MDS Clustering for 110th U.S. Senate",
        panel.grid.major=theme_blank())+
    xlab("")+ylab("")+scale_shape(name="Party", breaks=c("100","200","328"),
        labels=c("Dem.", "Rep.", "Ind."), solid=FALSE)+
    scale_color_manual(name="Party", values=c("100"="black","200"="dimgray",
                        "328"="grey"),
     breaks=c("100","200","328"), labels=c("Dem.", "Rep.", "Ind."))


print(base.110+geom_point(aes(shape=party, alpha=0.75, size=2)))
print(base.110+geom_text(aes(color=party, alpha=0.75, label=cong.110$name, size=2)))
```

现在，你应该很熟悉ggplot所做的很多事情了。可是，在创建这个实例的图形时，方式稍有不同。典型的过程是，首先创建一个ggplot对象，然后添加一个geom或者stat层，但是在这个实例中，我们创建了一个叫做case.110的基本对象，它包含了用于作图的所有格式化特征。这些特征包括大小（size）、阿尔法（alpha）、形状（shape）、颜色（color）和参数（opts）层。

之所以这样做是因为我们想要画两个图：第一个图，以不同党派对应不同的形状代表数据点画图；第二个图，以参议员的名字代替代表数据点，并且不同的党派用不同的颜色来画图。通过首先添加所有这些格式化层到base.110对象，然后就可以简单地增加sgeom_point层或者geom_text层到base对象上，以获得想要的图形。图9-4展示了这些画图结果。



图9-4：第110届美国国会记名投票的MDS聚类：a）以党派进行参议员聚类；b）以名字进行参议员聚类

首先来解答一开始所提问题：在使用记名投票记录进行聚类时，不同党派的参议员能否聚类到一起?从图9-4来看，答案显然是不能。在民主党人和共和党人之间有一个相当大的空白。这份数据也验证了一个事实，那就是人们通常认为的：这些参议员的对立相当明显。我们可以看看参议员Sanders，福蒙特州的独立参议员，他的位置非常靠左，而参议员Coburn和DeMint的位置相当靠右。同样明显的是，参议员Collins和Snowe在第110届国会靠近中间位置。在美国参议院的最近大多数大型立法战中，正是这些温和的共和党参议员成为中心人物。

另一个有趣的分析结果是参议员Obama和McCain在第110届国会的位置。Obama单独出现在图中的左上角，而McCain和参议员Wicker和Thomas聚集到一起，并且离中心更近。尽管这个结果的一个合理解释是，Obama和McCain在投票上具有非常强的互补性，但是我们知道数据背后的故事，更可能是因为这两位参议员由于总统竞选导致了他们在很多投票中同时缺席。也就是说，当他们对同一个法案进行投票时，他们也许已经具有了相对不同的投票习惯，可是投票记录的区别不是特别大，但是他们缺席的国会，通常会对同一个立法投票。当然，这引出了另一个问题：如何解释Wicker和Thomas的位置呢?

对于最后的可视化来说，我们将研究所有国会按时间排序的MDS图。这样做应该能给我们一些启示：随着时间变化，参议员整体上以党派聚集，而这也将使我们以更有说服力

的证据声明：现在参议员比以往更加两极分化。在前面的代码块中，我们通过do.call和rbind把rollcall.mds压缩进一个数据框，为所有的数据画了一个单独的图。接下来将要画出和前面步骤得到的一样的图，不过增加了一个facet_wrap，以便按时间顺序展示不同时期国会的MDS图。图9-5展示了这个可视化结果。

```
all.mds <- do.call(rbind, rollcall.mds)
all.plot <- ggplot(all.mds, aes(x=x, y=y))+
    geom_point(aes(shape=party, alpha=0.75, size=2))+
    scale_size(to=c(2,2), legend=FALSE)+
    scale_alpha(legend=FALSE)+theme_bw()+
    opts(axis.ticks=theme_blank(), axis.text.x=theme_blank(),
         axis.text.y=theme_blank(),
         title="Roll Call Vote MDS Clustering for U.S. Senate
         (101st - 111th Congress)",
         panel.grid.major=theme_blank())+
    xlab("")+ylab("")+
    scale_shape(name="Party", breaks=c("100","200","328"),
         labels=c("Dem.", "Rep.", "Ind."),
         solid=FALSE)+facet_wrap(~ congress)
all.plot
```

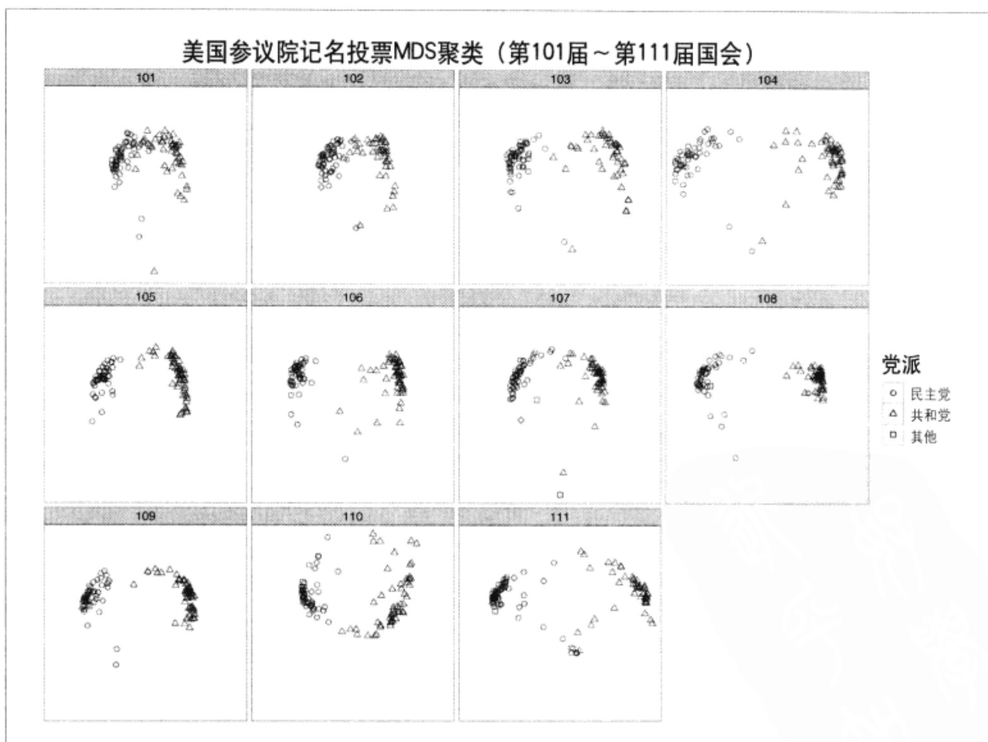


图9-5：美国参议院记名投票MDS聚类（第101届～第111届国会）

以记名投票作为参议院之间区别的度量标准，从这些结果中可以看出，美国参议院实际上和过去一样具有党派性。大体上来说，在每一届国会中我们只能看到大量的三角和圆形分别聚集到一起，而只有很少的异常数据点。你也许会说，第101届国会和第102届国会两极分化不那么严重，因为两个聚类看上去离得很近。但这是坐标轴刻度造成的结果。回忆一下，MDS程序只是简单地基于计算所得的所有观察值之间的距离矩阵，尝试去最小化一个损失函数。仅仅是因为第101届国会和第102届国会的作图刻度比其他很多届国会都小，这并不意味着这两届国会的两极分化更轻。这种差异可能是由很多原因造成的，例如观察值的数量。然而，因为我们在一个单独的图中对它们进行了可视化，必须在不同的面板中使用相同的刻度，所以这会导致有些图看上去更拥挤，而有些图铺得更开。

从图9-5中可以得到的重要结论是，在根据记名投票对参议员进行聚类时，不同党派聚集到一起的情况非常少。正如通过圆圈和三角聚类中分别存在的层次性看到的那样，尽管在党派内也许有轻微的变化，但是在党派之间的变化非常少。几乎在所有实例中，我们看到共和党人和共和党人聚类到一起，而民主党人和民主党人聚类到一起。当然，还有除了党派之外的其他信息，我们可以把感兴趣的这类信息添加到这个图中。例如，我们也许想知道，是否来自同一地区的参议员会被聚类到一起；或者，是否不同委员会成员会形成聚类。这些都是有趣的问题，建议读者超越当前这个初步的分析，深入挖掘这些数据。

第10章

kNN：推荐系统

k近邻算法

第9章介绍了如何利用简单的相关性技术，根据国会议员的投票记录来度量他们之间的相似程度。本章将会讨论如何利用同样的相似度度量方法来为网站用户进行推荐。

本章将讨论的算法称为*k*近邻（k-Nearest Neighbors，kNN）。它也许是本书中所有机器学习算法中最浅显易懂的一个了。如果有人想根据相似性推荐一些东西，几乎大部分人都会自动选择k近邻算法的最简单的版本：他们会为用户推荐一首与其喜欢的歌曲最相似的，但是他还未曾听过的歌。这种做法其实是1近邻算法。完整的kNN算法是这种直觉做法的一种扩展，在做出推荐之前，你可能会同时参考多于1个数据点。

完整的*k*近邻算法原理和我们向朋友们征求意见的原理差不多。首先，我们找到一些和我们品味相似的朋友，然后向他们征求意见。如果他们中大多数推荐了同样的东西，我们猜测这应该也是我们喜欢的东西。

我们该如何把上面的直观想法转换成一个可行的算法呢？在真实数据上进行推荐之前，让我们先从简单的情形开始：给点进行二分类。如果你还记得第3章中首次提到的分类问题，就不会对图10-1感到陌生。

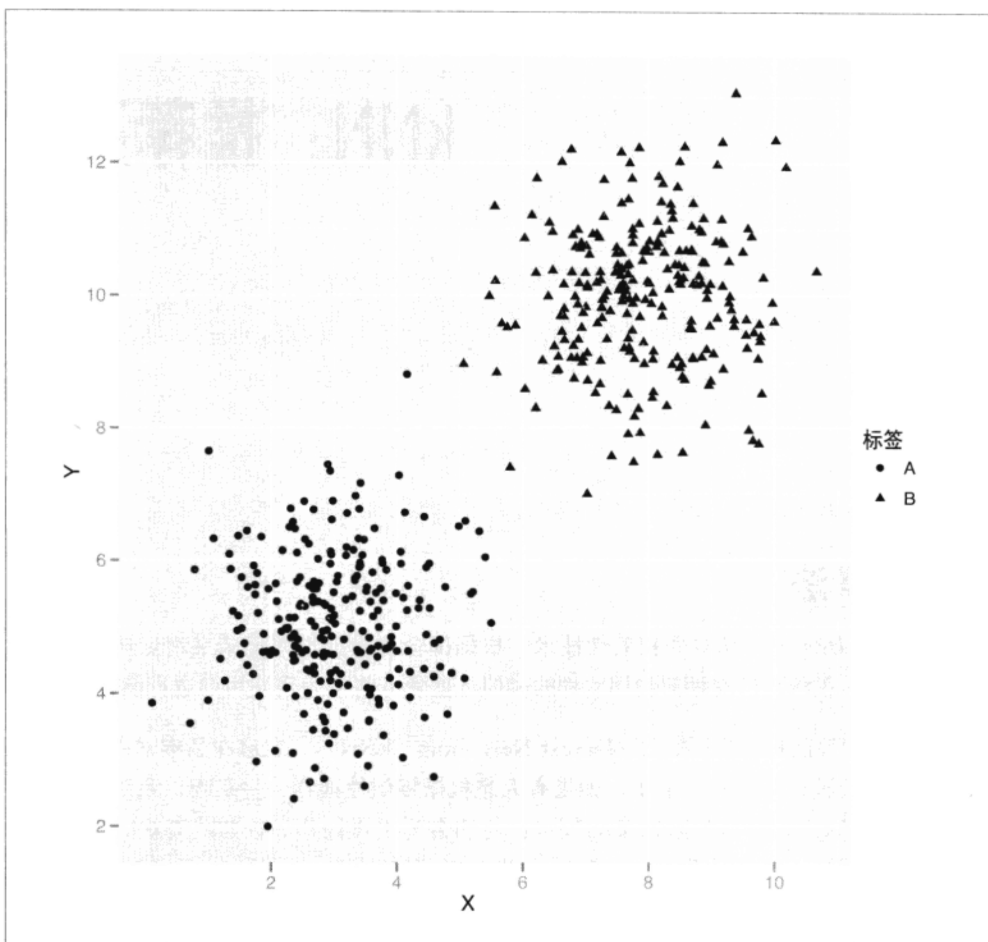


图10-1：线性决策边界的分类问题

正如我们当时解释的那样，你可以用glm函数对数据点进行逻辑回归，生成一条称为决策边界（decision boundary）的直线。在介绍逻辑回归的同时，我们也说了，类似图10-2那样的问题是不能通过一条简单的线性决策边界来解决的。

面对图10-2那样的有复杂决策边界的问题，你该如何构造一个分类器呢？你可以尝试使用非线性的方法，例如将在第12章中讨论的核方法（kernel trick）。

另外一种方法是利用目标点周围点的信息来帮助做分类。例如，你可以在目标点周围画一个圈，然后根据圈内点的情况来对目标点进行分类。图10-3展示了这种称为“草根民主”（Grassroots Democracy）算法的一个例子。

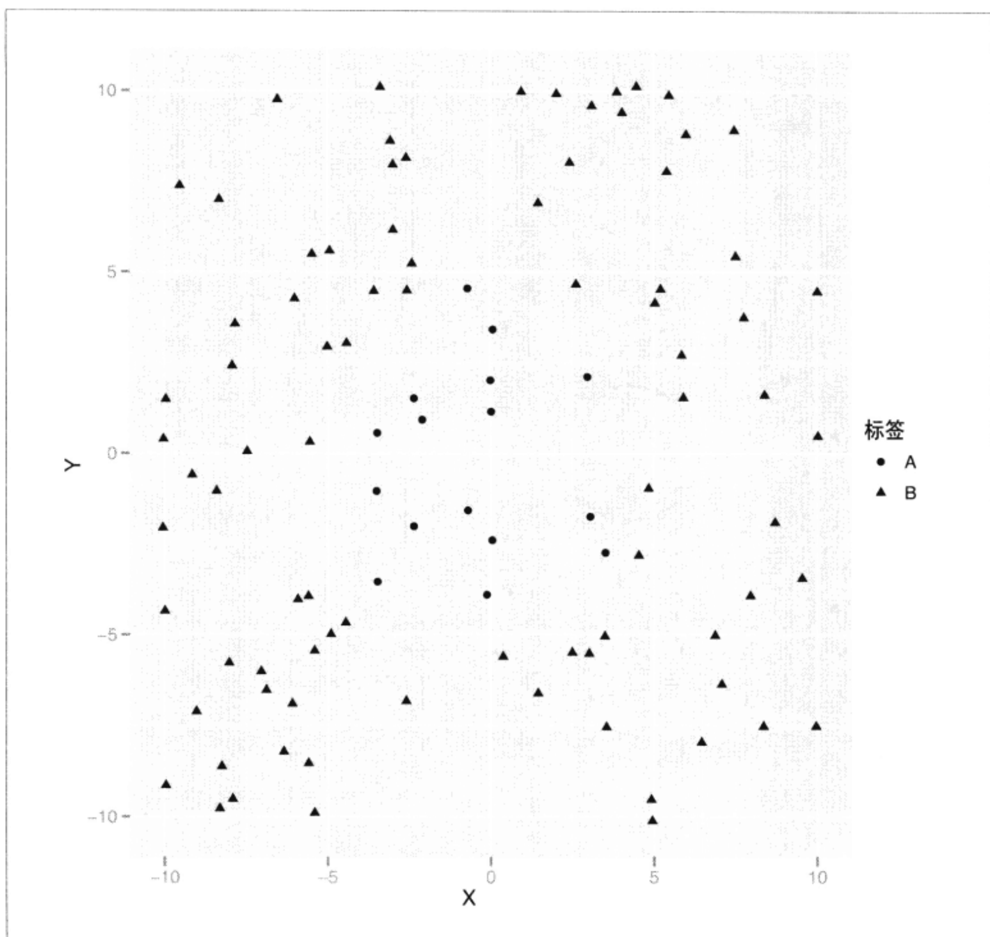


图10-2：非线性决策边界的分类问题

这个算法很有用，不过它有个很明显的缺陷：为了定义“附近”的点，我们必须为所画的圈定一个半径。如果所有的数据点之间的距离都差不多，那问题还不大。而如果有些数据点之间的距离非常近，而另一些数据点之间的距离非常远，我们就不得不选择一个非常大的半径画圈，以致对某些数据点的分类效果非常差。我们该怎么解决这个问题？比较明显的一个办法就是不再使用一个圆来圈定一个点的邻居，而改成参照与它最近的*k*个点。这些点就称为*k*近邻。一旦我们找到了*k*近邻，分析它们都是属于什么类别的，然后根据少数服从多数原则来为目标点定义类别。让我们把上面的思路用代码写出来。

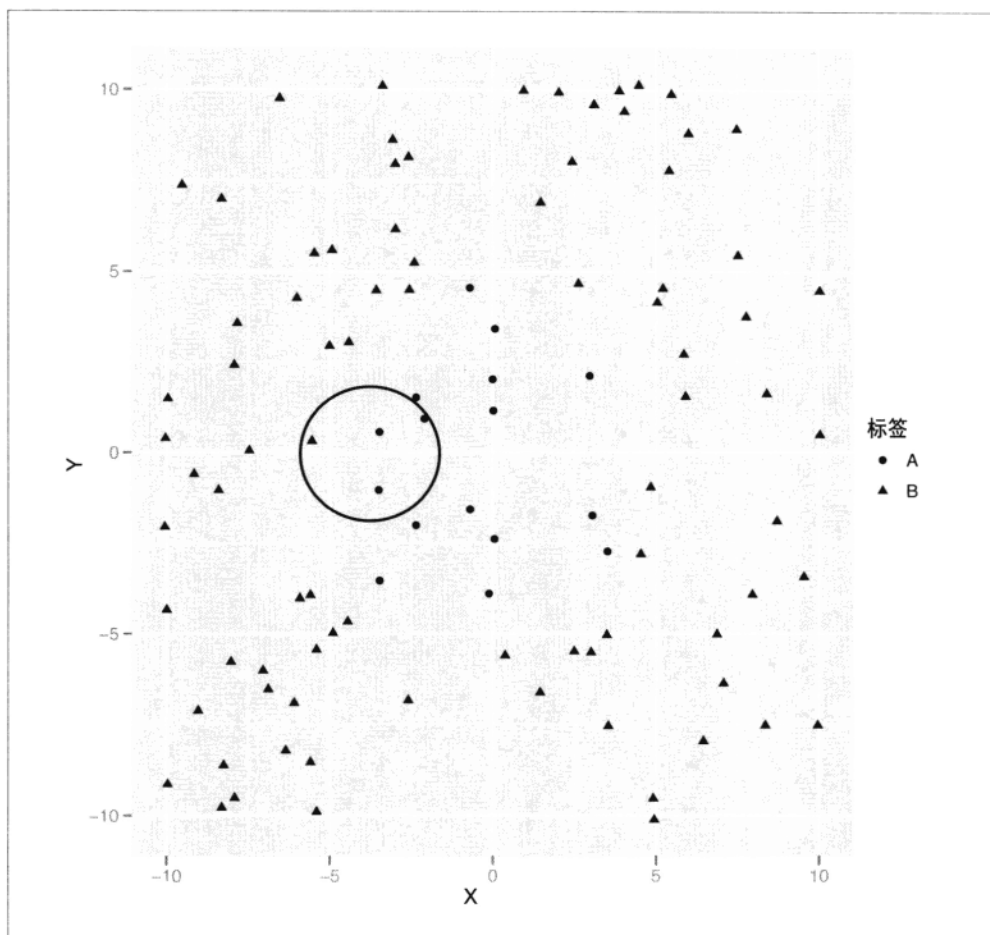


图10-3："草根民主"算法

首先，加载数据集：

```
df <- read.csv('data/example_data.csv')

head(df)
#          X        Y Label
#1 2.373546 5.398106     0
#2 3.183643 4.387974     0
#3 2.164371 5.341120     0
#4 4.595281 3.870637     0
#5 3.329508 6.433024     0
#6 2.179532 6.980400     0
```

接下来，我们要计算数据集之中所有点两两之间的距离，并把它们保存在距离矩阵里

面，其中数据点i和数据点j之间的距离保存在distance.matrix[i, j]里面。下面的代码使用欧式距离生成了这个距离矩阵：

```
distance.matrix <- function(df)
{
  distance <- matrix(rep(NA, nrow(df) ^ 2), nrow = nrow(df))

  for (i in 1:nrow(df))
  {
    for (j in 1:nrow(df))
    {
      distance[i, j] <- sqrt((df[i, 'X'] - df[j, 'X']) ^ 2 + (df[i, 'Y'] - df[j, 'Y'])
                             ^ 2)
    }
  }

  return(distance)
}
```

生成了距离矩阵之后，我们需要一个函数来返回某一个点的k近邻。这并不困难，假设你想要查到第i个数据点的k近邻，你可以取出距离矩阵的第i行，这样就可以得到其余每个点到第i个数据点的距离。把这一行排序之后，就得到了一组经过排序的点的列表，而排序依据是它们与第i个数据点之间的距离。这个排序结果列表的前k个点，除了输入点自身以外[译注1]，就是与数据点i最近的k个近邻。

```
k.nearest.neighbors <- function(i, distance, k = 5)
{
  return(order(distance[i, ])[2:(k + 1)])
}
```

上面这些工作都准备好了之后，我们会创建一个名为knn的函数，它包含两个输入，分别是一个数据框和参数k，这个函数会为数据框中的每一个点做出预测并返回。我们的函数并不通用，这是因为在这里假定分类标签都保存在一个叫做Label的列里面，不过这没关系，它只是让我们快速理解k近邻到底是怎么应用的。在R语言里面，有现成的k近邻实现可用于真实的应用，我们马上就会讨论到它。

```
knn <- function(df, k = 5)
{
  distance <- distance.matrix(df)

  predictions <- rep(NA, nrow(df))

  for (i in 1:nrow(df))
  {
     indices <- k.nearest.neighbors(i, distance, k = k)
```

译注1：总是这个排序列表的第一个，因为自己和自己的距离始终为0。

```
    predictions[i] <- ifelse(mean(df[indices, 'Label']) > 0.5, 1, 0)
  }

  return(predictions)
}
```

正如你所看到的，我们把k个近邻的分类标签值求平均，根据平均值是否大于0.5来给出预测，这遵循了一个少数服从多数的投票原则。调用这个函数会返回由预测结果组成的向量，我们可以把数据框作为参数传进去，然后评估一下分类效果：

```
df <- transform(df, kNNPredictions = knn(df))

sum(with(df, Label != kNNPredictions))
#[1] 7

nrow(df)
#[1] 100
```

在100个数据点之中，我们预测错了7个点，也是说准确率是93%。对于这么简单的一个算法来说，这还算不错。我们已经解释了k近邻算法是怎么工作的，接下来将k近邻算法应用在一些真实的数据上，比如关于R语言程序包使用情况的数据。

不过在开始之前，先展示一下如何在你自己的项目中使用R语言的k近邻实现：

```
rm('knn') # In case you still have our implementation in memory.
library('class')

df <- read.csv('data/example_data.csv')

n <- nrow(df)

set.seed(1)
indices <- sort(sample(1:n, n * (1 / 2)))

training.x <- df[indices, 1:2]
test.x <- df[-indices, 1:2]
training.y <- df[indices, 3]
test.y <- df[-indices, 3]

predicted.y <- knn(training.x, test.x, training.y, k = 5)
sum(predicted.y != test.y)

#[1] 7

length(test.y)
#[1] 50
```

在这里，我们使用了交叉验证测试了`class`程序包中的`knn`函数。令人意外的是，我们同样预测错了7个点，不过这次我们的测试集只有50，所以我们的准确率就只有86%了，不过，这也算不错了。为了做个对比，让我们看看逻辑回归模型的表现：

```
logit.model <- glm(Label ~ X + Y, data = df[indices, ])
predictions <- as.numeric(predict(logit.model, newdata = df[-indices, ]) > 0)
sum(predictions != test.y)
#[1] 16
```

正如你所看到的，最好的逻辑回归模型分错了16个点，准确率只有68%。当你的问题完全不是线性的时候，*k*近邻的表现比其他[译注2]方法要好得多。

R语言程序包安装数据

现在，我们已经大致了解了如何使用kNN算法，接下来看看如何利用kNN算法进行推荐。一种推荐的方法是，利用kNN算法为目标用户推荐那些与他已经喜欢的物品相似的物品，这种方法为基于物品（item-based）的方法。另一种方法是，我们先利用kNN算法找到与目标用户品味比较相近的用户，然后根据这些品味相近的用户的喜好来为目标用户进行推荐，这种方法称为基于用户（user-based）的方法。

两种方法听上去都比较合理，不过在一个具体的应用中，总会有某一种方法更适合。如果你的用户比物品多（就像Netflix网站上的用户数比电影数多一样），使用基于物品的推荐方法会节省很多的计算时间与存储空间，因为你只需要计算物品之间的两两相似度就可以了。如果你的物品比用户多（比如，当你刚刚开始收集用户数据的时候），那么你最好使用基于用户的推荐方法。本章将会关注基于物品的推荐方法。

不过，在讨论推荐系统的细节算法之前，让我们看一下打算进行推荐的应用场景和数据。我们使用的数据是在Kaggle网站（http://www.kaggle.com/）上进行的R程序包推荐竞赛中，为参赛者提供的数据。这份数据包括大约50名R语言程序员安装所有R程序包的信息。这个数据集并不大，不过足够用来分析不同程序包的流行程度以及它们之间的相似度。

注意： 竞赛中的获胜者通常都使用了kNN算法，尽管只是把它作为整个推荐算法的一部分。获胜者经常使用的另外一个算法叫做矩阵分解（matrix factorization）。关于矩阵分解算法，我们并不详细展开，不过如果你想搭建一个能够实际应用的工业级的推荐系统，需要考虑将kNN算法和矩阵分解模型结合起来使用。事实上，最好的系统通常将kNN、矩阵分解和其他分类器组合在一起生成一个超级模型（super-model）。将多个分类器组合起来的一些技巧通常称为集成方法（ensemble method）。

在这个R程序包推荐竞赛中，参赛选手需要根据一个程序员已经安装的程序包信息来预测这个程序员是否会安装另一个程序包。在预测一个程序员是否会安装新的程序包时，

译注2：这里的其他方法是指线性方法。

你需要看这个程序员是否安装了与这个程序包类似的其他程序包。换句话说，你会很自然地使用基于物品的kNN方法。

那么，让我们先把数据加载进来看一下，再为程序包之间的相似度做一个定义。因为为竞赛准备的原始数据比较复杂，所以我们将数据简化了一下，数据格式是这样的：每一行表示一个“用户－程序包”对，第三列表示这个用户是否安装了这个程序包。

```
installations <- read.csv('data/installations.csv')
head(installations)

#             Package User Installed
#1              abind    1         1
#2 AcceptanceSampling    1         0
#3             ACCLMA    1         0
#4           accuracy    1         1
#5            acepack    1         0
#6        aCGH.Spline    1         0
```

正如你所看到的，用户1安装了abind程序包，但是没有安装AcceptanceSampling程序包。我们把这份原始数据的格式转换成另外一种格式，就可以得到程序包之间的相似度的度量。我们将要把目前的“长数据格式”（long form）转换成一种“宽数据格式”（wide form），其中每一行对应一个用户，而每一列对应一个程序包。上述转换可以使用reshape程序包中的cast函数来实现。

```
library('reshape')
user.package.matrix <- cast(installations, User ~ Package, value = 'Installed')

user.package.matrix[, 1]
# [1]  1  3  4  5  6  7  8  9 11 13 14 15 16 19 21 23 25 26 27 28 29 30 31 33 34
#[26] 35 36 37 40 41 42 43 44 45 46 47 48 49 50 51 54 55 56 57 58 59 60 61 62 63
#[51] 64 65
user.package.matrix[, 2]

# [1] 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1
#[39] 1 1 1 1 1 1 1 1 0 1 1 1 1 1

row.names(user.package.matrix) <- user.package.matrix[, 1]
user.package.matrix <- user.package.matrix[, -1]
```

首先，我们使用cast函数来创建用户－程序包矩阵。我们发现，矩阵的第一列保存的仅仅是用户的ID，因此把它们保存在矩阵的row.names变量之后就把第一列删除。用现在的用户－程序包矩阵计算程序包相似度就非常容易了。为简单起见，我们使用列之间的相关性来衡量程序包之间的相似度。我们可以使用cor函数来计算相关性。

```
similarities <- cor(user.package.matrix)

nrow(similarities)
#[1] 2487
```

```
ncol(similarities)
#[1] 2487

similarities[1, 1]
#[1] 1
similarities[1, 2]
#[1] -0.04822428
```

现在，我们可以计算所有两两程序包之间的相似度了。如你所见，程序包1与它自己完全相似，与程序包2却不那么相似。不过，kNN使用的是距离，而不是相似度。因此，我们需要把相似度转换成距离。这里我们通过一些巧妙的数学技巧来把相似度1转换成距离0，而把相似度－1转换成距离无穷大。下面的代码实现了这个目的，如果它们不那么直观，请花一点时间仔细思考一下。

```
distances <- -log((similarities / 2) + 0.5)
```

完成上面的距离度量函数后，我们就可以开始实现kNN算法了。在这里，我们使用*k*=25，不过你最好使用不同的*k*来看看哪个*k*值对应的推荐效果最好。

为了进行推荐，我们假设，如果一个程序包的邻居程序包被安装的越多，那么它就越有可能被用户安装。接着我们根据一个程序包的邻居中已经被安装的个数来给程序包排序，将排序最靠前的程序包推荐给用户。

因此，我们来实现k.nearest.neighbors函数：

```
k.nearest.neighbors <- function(i, distances, k = 25)
{
  return(order(distances[i, ])[2:(k + 1)])
}
```

使用最近邻居的信息，我们根据它的邻居有多少个已经被安装来预测它会被安装的概率：

```
installation.probability <- function(user, package, user.package.matrix, distances, k = 25)
{
  neighbors <- k.nearest.neighbors(package, distances, k = k)

  return(mean(sapply(neighbors, function (neighbor) {user.package.matrix[user,
              neighbor]})))
}

installation.probability(1, 1, user.package.matrix, distances)
#[1] 0.76
```

从上面的代码可以看出，对于用户1来说，他有0.76的概率可能安装了程序包1。因此，

我们要做的是，找到用户最可能安装的程序包，再把它推荐给用户。我们通过循环遍历所有的程序包，分别计算出它们被安装的概率，再给出最有可能的那个：

```
most.probable.packages <- function(user, user.package.matrix, distances, k = 25)
{
  return(order(sapply(1:ncol(user.package.matrix),
                      function (package)
                      {
                        installation.probability(user,
                                                 package,
                                                 user.package.matrix,
                                                 distances,
                                                 k = k)
                       }),
                decreasing = TRUE))
}

user <- 1
listing <- most.probable.packages(user, user.package.matrix, distances)
colnames(user.package.matrix)[listing[1:10]]
#[1] "adegenet"     "AIGIS"        "ConvergenceConcepts"
#[4] "corcounts"    "DBI"          "DSpat"
#[7] "ecodist"      "eiPack"       "envelope"
#[10]"fBasics"
```

这种推荐方法的一大好处在于，它是可解释的。我们可以说，之所以推荐程序包P给用户，是因为他已经安装了程序包X、Y和Z。这种可解释性在有些应用场景中是非常有用的。

我们仅仅使用相似性度量就构建了一个推荐系统。在第11章中，我们将深入地利用社交网络的信息来构建一个推荐引擎。

第11章

分析社交图谱

社交网络分析

社交网络无处不在。根据维基百科（Wikipedia）的数据，在互联网上有超过200个活跃的社交网站，其中还不包括那些交友网站。从图11-1可以看出，根据Google趋势（Google Trends）的数据，从2005年开始，全球对于“社交网络”的兴趣一直在稳定增长。这种情况是非常合理的：对于社会交往的渴望，是人类最基本的天性，而这种社会本质必然会体现在我们的技术中，就显得不足为奇了。但是关于社交网络的映射和建模，并没有什么新东西出现。

在数学界，一个社交网络分析的例子是，计算一个人的埃尔德什（Erdős）数，它衡量这个人与著名的高产数学家保罗·埃尔德什之间的距离。埃尔德什是20世纪无可争议的最高产数学家，他在职业生涯中一共发表了1500多篇论文。这些论文很多是与其他人合著的，而埃尔德什数衡量了一位数学家到埃尔德什的合著者圈子的距离。如果一位数学家与埃尔德什合著过一篇论文，那么他的埃尔德什数是1，也就是说他在由20世纪数学家组成的网络中，到埃尔德什的距离是1。如果另外一个作者和埃尔德什的合著者合作过，但是没有和埃尔德什本人直接合作过，那么这位作者的埃尔德什数就是2，以此类推。尽管这个衡量标准不是特别严谨，但是它粗略地反映了一个人在数学界的声望。埃尔德什数使我们能够迅速获得围绕保罗·埃尔德什的庞大数学家网络。

欧文·戈夫曼（Erving Goffman）是20世纪最著名的知识分子之一，他对社会科学有巨大贡献，堪称社会学界的保罗·埃尔德什。关于人类的交往天性，他的评论是最佳的评论之一：

当人们在别人面前时，他们能发挥的就不仅仅是身体功能了，同时也能发挥交流功能。这种发挥交流功能的机会不比发挥物理功能的机会低，也是每个人都关注的重要事情，而它在每种社会形态中，都处在严格的标准规范下，从而产生了一种沟通的交通秩序。

——欧文·戈夫曼

《公共场所行为：关于社会组织聚集的说明》（1996）

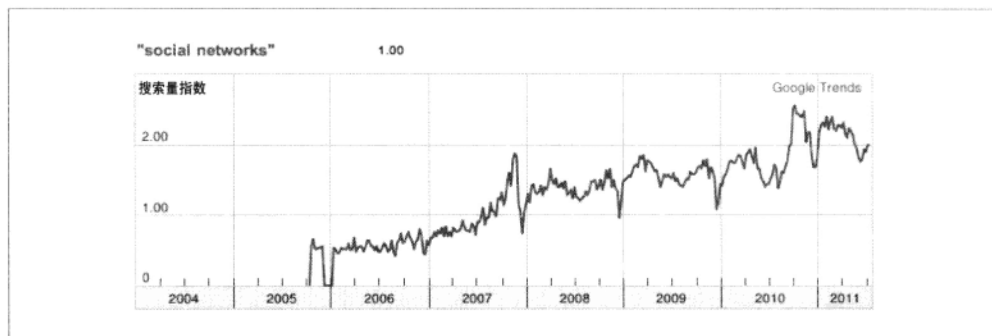


图11-1：Google搜索中Social networks（社交网络）的增长

戈夫曼提到的“交通秩序”指的就是社交网络。人们渴望彼此交往并适应社会，这种渴望的副产品就是彼此之间形成了结构性很强的图，它提供了一种由人们的身份标识到历史行为记录的“映射”。像Facebook、Twitter和LinkedIn这些社交网络服务，它们只是提供了用于进行“交往”这种人类最本质行为的高度程式化的模板。这些服务的创新之处并不是它们的功能，而是提供了洞察大部分人类社交关系图的方法。对于像我们这样的黑客来说，社交网络站点暴露出来的数据就是名副其实的美妙甘泉。

但是社交关系图的价值并不局限于社交网络站点。有其他几种关系也可以建模成一个网络，并且它们中大多数的数据也都可以通过各种各样的互联网服务获得。例如，我们可以基于观众在Netflix上看过的电影，在他们之间建立关系映射。同理，基于听众使用像Last.fm或者Spotify这样的音乐服务的行为模式，我们可以演示不同音乐类型之间是如何关联的。我们也可以用一个更基本的方式把一个计算机局域网，甚至整个互联网的结构，都用一大堆点和边来建模。

尽管得益于社交网络站点的传播，现在对社交网络的研究非常流行，但是一般所说的“社交网络分析”，指的是在过去几十年中一直被使用和发展的工具集合。它的核心思想是，对于各种网络的研究，都依赖于使用图论的语言，去描述有内在联系的对象。早在1736年，欧拉（Euler）就使用了点和边的概念，公式化地描述柯尼斯堡桥（Königsberg Bridge）问题。

注意： 柯尼斯堡桥问题，是旅行商问题的一个早期变型，它要求设计一条穿过东普鲁士柯尼斯堡（现在的俄罗斯加里宁格勒）的路径，必须满足在每一座桥都只能走一遍的前提下，把这个地方所有的7座桥都走遍。欧拉把这座城市的地图转化成一个有4个点（城市的区域）和7条边（7座桥）的简单图，并且解决了这个问题。

早在20世纪20年代，著名的心理学家Jacob L. Moreno开发了研究人类关系的一种方法，这种方法叫做“计量社会学”（sociometry）。Moreno感兴趣的是，人们的社会交往如何影响他们的幸福感，因此他询问人们都有哪些朋友，由此开始映射这种关系结构。在1937年，人类学家Joan Criswell使用Moreno的计量社会学方法研究白人和黑人初中生之间的种族分歧[JC37]。

我们认为，现代社交网络分析是多种学科理论和方法的聚合产物。这些理论和方法很大一部分来自社会学，包括Linton Freeman、Harrison White、Mark Granovetter以及其他许多杰出学者都作出了巨大贡献。类似，许多贡献也来自于物理学、经济学、计算机科学、政治学、心理学以及无法一一列举的其他学科和学者。在这里无法列出太多的作者和引文，而且有很多书籍著作综述了这个巨大研究领域的各种方法。其中一些最容易理解的书籍包括：《社会网络分析：方法与应用》（Social Network Analysis，Stanley Wasserman和Katherine Faust著[WF94]）、《社会与经济网络》（Social and Economic Networks，Matthew O. Jackson著[MJ10]）以及《网络、增长及市场》（Networks, Crowds, and Markets，David Easley和Jon Kleinberg著[EK10]）。在这个社交网络的简单介绍当中，我们只涉及了这个主题的很小一部分。对于那些有兴趣学习更多知识的读者来说，强烈推荐阅读上面提到的任何一本著作。

那么我们将在本章中涉及哪些内容呢？按照本书的一贯做法，我们将专注于一个社交网络的案例研究，它将带着我们经历整个数据研究周期，包括：获取社交网络数据；对数据进行清理和结构化处理；最后分析数据。在这个案例中，我们专注于当今杰出的“公开”社交网络：Twitter。“公开”是带引号的，这是因为Twitter并没有真的公开到允许我们随意访问它所有数据的程度。和其他许多社交网络一样，它提供了一个带有严格访问频率限制的API（应用程序编程接口）。因此，我们将要构建一个用于从Twitter提取数据的系统，它既不会超过这个访问频率限制，也不会违反Twitter的服务条款。事实上，我们在本章中将不会直接访问Twitter的API。

我们的项目以构建一个本地网络或者称为个体网络（ego-network）开始，并且使用与计算埃尔德什数同样的方法，从这个起点开始，像滚雪球一样扩展。就首次分析而言，我们将研究圈子发现的方法，这些方法试图将社交网络分割成凝聚圈子（cohesive subgroup）。在Twitter中，可以通过这种分析知道一个用户属于哪些社交圈子。最后，因为本书是关于机器学习的，所以我们将使用Twitter的社交关系图来建立一个关于“可

能感兴趣的人”的推荐引擎。

我们会避免使用难以理解的专业术语去描述正在做的事情。可是，为了完成本章的目标，有一些术语值得去学会使用。我们只介绍了“个体网络”这一个术语，它一般用于描述一种图[译注1]。因为我们接下来将会多次使用“个体网络”，所以有必要清楚定义这个术语。“个体网络”是指在网络中直接包围在一个单独节点周围的社交关系结构。具体来说，“个体网络”是一个网络的子集，它包括：一个种子（个体）和它的邻居，也就是直接与种子相连的那些节点，连接种子和这些点的边，以及这些邻居之间连接的边。

以图的方式进行思考

在深入分析Twitter的社交关系图之前，先退一步，对网络下个定义，这有利于下一步工作。在数学中，一个网络（或者叫图）只不过是一组点和边的集合。这些集合没有任何上下文含义，它们只是为了呈现一些边把一个个点连接起来形成的区域。最抽象的公式化描述是：那些边除了把两个点连接起来之外，不包含任何信息。不过，接下来我们得看到这种常见的数据组织方式如何迅速变得复杂化。考虑图11-2中的三个图片。

图11-2a是一个无向图的例子（undirected graph）。在这种情况下，图的边是没有方向的。换一种方式考虑这种情况就是：点之间的边暗示我们点的连接关系是双向的。例如，Dick和Harry共享一个连接，因为这个图是无向的，所以我们可以设想这意味着他们彼此之间可以通过这个联系互换信息、食物等。另外，因为这是最基本的网络，所以很容忽略这样一个事实：即使用的是无向图，我们也已经在前面提到的最一般的图的基础上增加了复杂性。在图11-2所示的所有图中，我们为每个点都增加了标签。虽然这是一个额外添加的信息，但是它在这里非常有用，当我们考虑如何从一个网络抽象映射到对实际数据的描述时，它是非常重要的。

继续看图11-2b，这是一个有向图。在这个图中，每条边都有一个表明它的方向的箭头。现在边不再是双向的了，而是表示了一种单向关系。例如，Dick和Drew都有一个到John的联系，而John只有一个到Harry的联系。在图11-2c，图中增加了“边标签”。特别是，我们为每个边都增加了一个正号或者负号标签。它可以用来表示这个网络中成员之间的“喜欢”或者“不喜欢”关系，或者接近的某种程度。和点标签一样，边标签也增加了上下文信息。这些标签也可以比现在这种简单的二元关系更加复杂，它们也可以是表示强度或者关系类型的权重值。

译注1：在本章中，图和网络是同一概念。

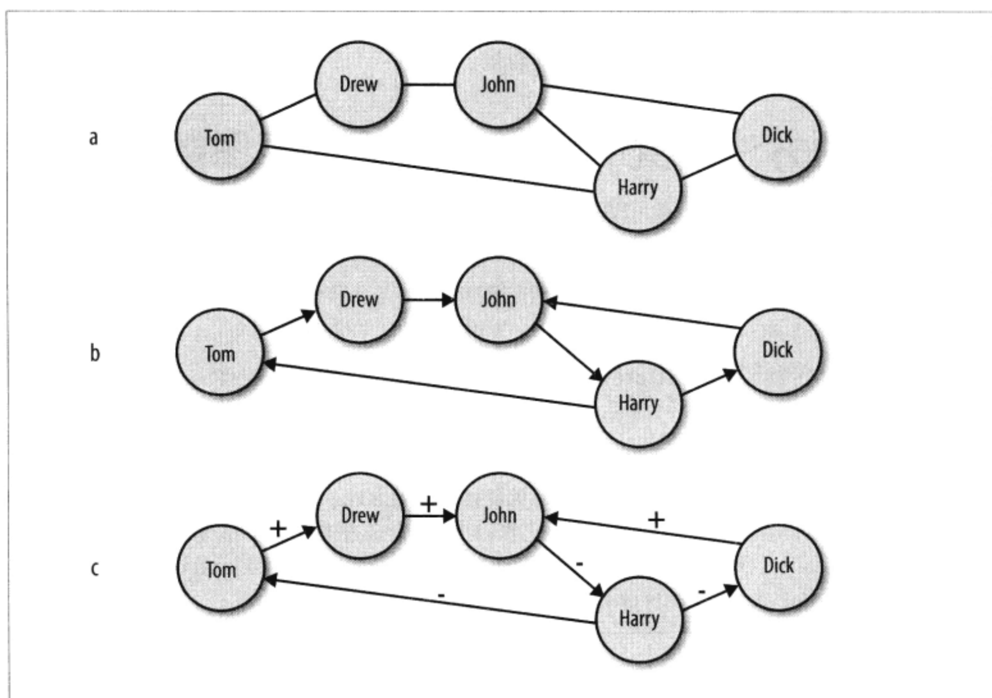


图11-2：不同类型的网络：a)无向图；b）有向图；c）带权有向图

因为我们在互联网上将会遇到很多不同形式的社交关系数据，所以上述各种图之间的区别就是需要重点考虑的了。图的不同结构可以影响人们使用相应服务的方式，因此也会影响我们分析相关数据的方式。考虑基于两种不同类型图结构的流行社交网络站点：Facebook和Twitter。Facebook是个大型的无向社交关系图。因为“好友关系”要求通过验证，所以所有的边都表示一个互为好友的关系。这个特点导致Facebook演化为一个拥有密集本地网络结构的相对更加封闭的社交网络，而不是一个巨大的开放服务中心。而Twitter是一个巨大的有向图。Twitter上的“关注”互动方式不需要互相验证，因此Twitter的无向图具有很强的不对称性，名人、新闻报道机构以及其他高级用户都是图中主要的信息广播点。

尽管上述两种互联网服务之间节点连接方式的区别看上去很细微，但是产生的结果却有巨大的差别。通过对互动方式进行细微改变，Twitter创建了一个完全不同与Facebook的社交关系图，同时人们使用Twitter与使用Facebook之间的方式也具有很大的不同。当然，产生这种不同的原因不只是因为图的结构，也包括Twitter的140个字符限制和大多数Twitter账户的完全公开本质，但是这个图的结构很大程度上影响了整个网络的运转方式。在本章的案例分析中，我们将专注于Twitter，因此需要在分析过程中的所有方面都

考虑它的有向图结构。首先面临的一个挑战是，在不超过API使用频率限制，且不违反使用条款的情况下，收集Twitter中的关系数据。

用黑客的方法研究Twitter的社交关系图数据

在写本章的时候，Twitter提供了两种不同访问形式的API。第一种是无身份验证的，它允许每小时进行150次请求。第二种是开放授权（OAuth）身份认证API，对于调用的限制是每小时350次。每小时150次API调用的限制实在是太严格了，以致我们无法在合理的时间内抓取到需要的数据。第二种350次的限制还是太严格了，而且，我们没有兴趣创建一个需要身份验证的Twitter应用。我们想要快速获得数据来构建一个网络，然后开始钻研这份数据。

注意： 如果你不熟悉表述性状态转移应用程序编程接口（RESTful API），或者没有听说过开放授权，请不要担心。对于这个例子来说，我们将不关注这些细节。如果你想要学习更多与它们相关的知识，我们建议你阅读相关服务的文档。对于Twitter来说，最好的参考资料是关于API的FAQ：*https://dev.twitter.com/docs/api-faq*。

遗憾的是，如果使用Twitter自己提供的API，我们将无法完成这个任务。为了获得想要的数据，我们将不得不使用另外一份Twitter社交关系图数据资源：Google社交关系图API（SocialGraph API，SGA）。Google在2008年引入SGA的目的是为了建立用于所有社交网络网站的统一身份识别。例如，你也许拥有几个在线服务账号，所有这些账号都可以解析成一个统一的电子邮件地址。SGA的想法是，使用这个统一账号信息来收集你跨多个服务的完整社交关系图。如果你很好奇Google中有多少关于你的数字信息，在任意网页浏览器中输入如下地址[译注2]：

```
https://socialgraph.googleapis.com/lookup?q=@EMAIL_ADDRESS&fme=1&pretty=1
```

如果你把EMAIL_ADDRESS替换成合适的电子邮件地址，这个API将在浏览器窗口中返回一个原始JSON数据结构，从中你可以看到Google已经存储了多少关于你的社交关系图。如果你有一个用上面输入的电子邮件地址注册的Twitter账号，你将会在返回数据中发现你的Twitter页面链接地址（URL），它是其中一个被解析为电子邮件地址的社交关系图服务。Twitter是SGA的众多公开社交关系图数据抓取目标之一。因此，我们可以使用SGA查询指定用户的社交网络，然后创建用于我们研究的网络。

使用SGA的最主要优势是，不像Twitter API那样每小时只提供很少量的查询，SGA每天允许50 000次查询。即使是在Twitter这个级别的数据，这也足够用于创建绝大多数用户

译注2：遗憾的是，Google已经关闭了这项服务。

的社交关系图了。当然，你如果你是艾什顿·库奇[译注3]（Ashton Kutcher）或者蒂姆·奥莱利[译注4]（Tim O'Reilly），这个案例研究中提到的方法将不起任何作用。也就是说，如果你是一位名人并且也对于做这个练习有兴趣，欢迎你使用我们的Twitter用户名：@johnmyleswhite或者@drewconway。

```
https://socialgraph.googleapis.com/lookup?q=http://twitter.com/
        drewconway&edo=1&edi=1&pretty=1
```

举例来说，我们利用Drew的Twitter页面来探索SGA如何组织数据结构。在你的网页浏览器中输入上面的API查询语句。如果你愿意，欢迎你使用自己的Twitter用户名查询SGA。和之前一样，SGA返回描述Drew Twitter关系的原始JSON数据结构。在API查询中，我们看到三个重要的参数：edo、edi和pretty。因为pretty参数用于返回格式化的JSON，所以它只在浏览器视图中有用。在实际解析这个JSON以生成网络的时候，我们将丢弃这个参数。不过edi和edo对应Twitter关系的方向。edo参数代表“出边”（edges out），也就是Twitter上的朋友，而edi的意思是“入边”（edges in），也就是Twitter上的粉丝。

在例11-1中，使用@drewconway 查询得到简化版的格式化原始JSON数据结构。我们最感兴趣的JSON对象是节点（nodes）。它包含关于Twitter用户的一些描述信息，但是其中更重要的是节点的“入边”和“出边”关系。nodes_referenced对象包含被查询节点的所有Twitter好友。他们是这个节点关注的其他用户（出度（out-degree））。类似，nodes_referenced_by对象包含了所有关注这个节点的用户（入度（in-chegree））。

例11-1：Google社交关系图的原始JSON数据结构

```
{
  "canonical_mapping": {
  "http://twitter.com/drewconway": "http://twitter.com/drewconway"
},
 "nodes": {
  "http://twitter.com/drewconway": {
  "attributes": {
    "exists": "1",
    "bio": "Hopeful academic, data nerd, average hacker, student of conflict.",
    "profile": "http://twitter.com/drewconway",
    "rss": "http://twitter.com/statuses/user_timeline/drewconway.rss",
    "atom": "http://twitter.com/statuses/user_timeline/drewconway.atom",
    "url": "http://twitter.com/drewconway"
  },
  "nodes_referenced": {
      ...
```

译注3：Ashton Kutcher是美国好莱坞男星。

译注4：Tim O'Reilly是O'Reilly公司创始人。

```
    },
    "nodes_referenced_by": {
      ...
    }
  }
}
```

警告： SGA爬虫只是扫描Twitter用户间的公开链接，并且将这种关系存放到它的数据库中。因此这份数据中不包括私有Twitter账号。如果你检查自己的SGA数据，也许会注意到，它并没有准确地反映你当前所有的Twitter关系。特别是，node_referenced对象也返回你不再关注的用户。这是因为SGA只会周期性地抓取活动链接，所以它的缓存数据并不总是反映最新的数据。

当然，我们并不想通过浏览器进行与SGA相关的工作。这个API原本就是用来被程序访问的，而它返回的JSON需要被解析成图对象，以便我们可以创建并且分析这些Twitter网络。为了创建这些图对象，我们的策略是使用一个单独的Twitter用户作为种子（回忆一下埃尔德什数的计算法方法），并且由这个种子为起点创建这个网络。我们将使用一个叫做滚雪球取样的方法，它以一个单独的种子为起点，然后找到所有连接这个种子的“入度”和“出度”。然后我们将使用那些连接节点作为新的种子，并且将这个过程重复一定次数。

对于这个案例研究来说，我们将会进行两轮取样，这样做有两个原因。首先，我们感兴趣的是映射并且分析这个种子用户的本地网络结构，以便给种子用户推荐他可能感兴趣的人。其次，因为Twitter社交关系图的规模，如果使用更多的轮次取样，我们也许会很快超过SGA的使用率限制和我们的硬盘存储空间。记住这些限制，在下一节中，我们将以开发一个使用SGA进行工作，并且解析数据的基本函数作为开始。

使用Google社交关系图API进行工作

Google社交关系图API的主要变化

在本书将要印刷的时候，我们注意到Google社交关系图API存储的Twitter数据量和我们完成本章的草稿时不再一样了。因此，如果你完全按照本节提供的代码运行程序，结果数据将会和完成本章案例研究所需要的数据有很大的不同。遗憾的是，我们对于解决这个数据问题无能为力。我们决定保留本节，是因为我们认为揭示如何使用API进行工作相当重要，而且我们也不想让读者错过这段讲解。我们在本书的补充文件中提供了一些样本数据集，它们是在Google社交关系图API发生变化之前，使用本节的代码收集的。你可以使用这份数据完成本章案例研究的剩余部分。

我们首先要做的是加载生成Twitter关系图所需的R语言程序包。为了从SGA创建这些图，我们将使用3个R语言程序包。RCurl程序主要提供了一个到libcurl库的接口，我们将使用它给SGA发送HTTP请求。然后，我们将使用RJSONIO程序包解析SGA返回的、存放在R语言列表容器内的JSON数据结构。碰巧这两个程序包都是由DuncanTemple Lang开发的，他已经开发了很多非常重要的R程序包（参见*http://www.omegahat.org/*）。最后，我们将使用igraph程序包创建和存储网络对象。igraph是一个强大的R语言程序包，用于创建和操作图对象，而且在开始使用网络数据进行工作时，它的灵活性对我们来说非常有价值。

```
library(RCurl)
library(RJSONIO)
library(igraph)
```

我们要写的第一个使用SGA工作的函数抽象级别最高。这个函数叫做`twitter.network`，它使用一个给定的种子用户查询SGA，解析JSON数据结构，并且返回一个对象来表示种子用户的个体网络。这个函数只有一个参数：`user`，它是一个表示Twitter种子用户的字符串。然后这个函数以GET模式创建相应的SGA请求URL，并且使用`RCurl`中的`getURL`函数执行HTTP请求。因为滚雪球搜索一次需要发送很多HTTP请求，所以我们创建了一个`while`循环，通过循环条件确保可以处理SGA服务不可用的情况。如果没有这个条件，这脚本将会忽略滚雪球搜索中的这些节点，但是有了这个检查，它就会回滚并且尝试重新请求，直到收集到相应的数据。一旦确认API请求返回了所需的JSON数据结构，就使用`RJSONIO`程序包中的`fromJSON`函数来解析它。

```
twitter.network <- function(user) {
    api.url <-
    paste("https://socialgraph.googleapis.com/lookup?q=http://twitter.com/",
                 user, "&edo=1&edi=1", sep="")
    api.get <- getURL(api.url)
    # To guard against web-request issues, we create this loop
    # to ensure we actually get something back from getURL.
    while(grepl("Service Unavailable. Please try again later.", api.get)) {
        api.get <- getURL(api.url)
    }
    api.json <- fromJSON(api.get)
    return(build.ego(api.json))
}
```

在解析JSON数据结构之后，我们需要用解析得到的数据创建网络。回想一下可知，数据中包含两种类型的关系：`node_referenced`（出度）和`nodes_referenced_by`（入度）。因此，为了得到两种类型的边，我们需要创建两个程序分支来处理数据。为了完成这个目标，我们定义了`build.ego`函数，它接受解析来自SGA的JSON数据得到的列表对象作为参数，并且创建网络。可是在这之前，我们必须清洗SGA返回的关系数据。如果仔细检查API请求返回的所有节点，你将会发现有些节点并不是Twitter用户。返回结果中包

含的这些关系和这个案例没有任何关系。例如，很多Twitter数据列表都有表示“账户”（account）重定向的URL。尽管这些都是Twitter社交关系图的一部分，但是我们不想在图中包含这些节点，因此需要定义一个辅助函数来删除它们。

```
find.twitter <- function(node.vector) {
    twitter.nodes <- node.vector[grepl("http://twitter.com/", node.vector,
                                      fixed=TRUE)]
        if(length(twitter.nodes) > 0) {
             twitter.users <- strsplit(twitter.nodes, "/")
           user.vec <- sapply(1:length(twitter.users),
                         function(i) (ifelse(twitter.users[[i]][4]=="account",
                                      NA, twitter.users[[i]][4])))
             return(user.vec[which(!is.na(user.vec))])
        }
    else {
             return(character(0))
    }
}
```

find.twittter函数用来执行数据清洗工作，它通过检查SGA返回的URL的结构来验证那些真正的Twitter用户，并删除社交关系图中的其他部分。这个函数做的第一件事是，通过使用grepl函数检查URL是否包括“http://twitter.com”模式，来核实SGA返回的节点确实来自Twitter。对于那些匹配了模式的URL列表来说，我们需要找出与实际账户对应的那一条URL，而不是整个URL列表或者重定向URL。完成这个任务的一个办法是，使用“反斜杠”（/）来分割URL列表，然后查找“account”字段，它表示一个非Twitter用户URL。识别出那些可以匹配非Twitter用户模式的URL后，我们只需要返回那些没有匹配这个模式的URL。这个函数的返回值将告诉build.ego函数哪些节点应该添加到网络中。

```
build.ego <- function(json) {
        # Find the Twitter user associated with the seed user
        ego <- find.twitter(names(json$nodes))
        # Build the in- and out-degree edgelist for the user
        nodes.out <- names(json$nodes[[1]]$nodes_referenced)
        if(length(nodes.out) > 0) {
             # No connections, at all
             twitter.friends <- find.twitter(nodes.out)
             if(length(twitter.friends) > 0) {
                     # No twitter connections
                     friends <- cbind(ego, twitter.friends)
             }
             else {
                     friends <- c(integer(0), integer(0))
             }
        }
        else {
             friends <- c(integer(0), integer(0))
        }
        nodes.in <- names(json$nodes[[1]]$nodes_referenced_by)
```

```
        if(length(nodes.in) > 0) {
            twitter.followers <- find.twitter(nodes.in)
            if(length(twitter.followers) > 0) {
                    followers <- cbind(twitter.followers, ego)
            }
            else {
                    followers <- c(integer(0), integer(0))
            }
        }
        else {
            followers <- c(integer(0), integer(0))
        }
    ego.el <- rbind(friends, followers)
    return(ego.el)
}
```

识别出列表中正确的节点之后，现在就可以开始构建网络了。为了完成这个任务，我们将使用build.ego函数来创建一个“边列表”，用它描述种子用户的出度和入度关系。一个“边列表”（edge list）是只有两列的简单矩阵，它用于表示有向图中的关系。第一列表示起始节点，而第二列表示终到节点。你可以理解为这种结构表示第一列中的节点连接到第二列中的节点。通常来说，“边列表”将包含那些是整数或者字符串类型的节点的标签。在这个例子中，我们将使用Twitter的用户名（字符串）作为标签。

尽管build.ego函数的代码很长，但是实际上它的功能很简单。它的大多数代码用来在构建“边列表”时进行数据组织和错误检查。你可以看到，首先检查的是调用结果中nodes_referenced和nodes_referenced_by是否都返回了一些关系数据。然后，我们检查哪些节点是实际的Twitter用户。如果这两个检查中的任何一个失败，那么返回一个特殊的向量作为函数的输出：c(integer(0), integer(0))。通过这个过程，我们创建了两个“边列表”矩阵，分别叫做好友（friends）和粉丝（followers）。最后一步，我们使用rbind函数来把这两个矩阵绑定到一个叫做ego.el的单独“边列表”中。在没有数据的情况下，我们使用了特殊的向量c(integer(0), integer(0))，因为rbind将会忽略它，所以结果不会受到影响。通过在R语言控制台中输入以下代码，你可以看到这一点：

```
rbind(c(1,2), c(integer(0), integer(0)))
     [,1] [,2]
[1,] 1 2
```

注意：你也许会好奇，为什么我们只是创建关系的“边列表”，而不是使用igraph直接创建图。因为igraph存储图的方式是放在内存中，所以很难像滚雪球取样要求的那样以迭代方式创建图。因此，更简单的办法是先创建整个关系数据集，然后再把这份数据转化成一个图。

现在我们已经完成了建图脚本的核心代码。build.ego函数完成了最难的那部分工作：接受经过解析的JSON数据结构，并把这些来自SGA的数据转换成可以通过igraph转化

为一个网络对象的数据集。我们需要完成的最后一部分代码，是利用所有函数一起实现滚雪球取样功能。我们将定义twitter.snowball函数来进行滚雪球取样，并且定义一个简单的辅助函数get.seeds，用它来产生需要访问的新种子列表。和build.ego函数一样，twitter.snowball函数的主要目的是产生关系网络的边列表。不过在使用twitter.snowball的时候，我们将把多次调用build.ego函数的结果绑定到一起，并且返回一个igraph图对象。为了简便起见，我们从定义get.seeds函数入手，尽管它将用到twitter.snowball函数中产生的数据。

```
get.seeds <- function(snowball.el, seed) {
    new.seeds <- unique(c(snowball.el[,1],snowball.el[,2]))
    return(new.seeds[which(new.seeds!=seed)])
}
```

get.seeds函数的目标是，从一个边列表中找到一组不重复的非种子节点作为新的种子节点集合。通过把由两列组成的边列表矩阵降维成一个向量，然后对向量中的元素进行去重，就可以很容易地完成获得候选种子的任务。去重得到的新向量叫做new.seeds，而且这个向量中只有不是种子的元素才会被函数返回。这是个简单而有效的方法，但是在使用它的时候有一点我们必须考虑。

get.seeds函数只会确保新种子和当前的种子不一样。在整个滚雪球取样的过程中，我们想要使用某些方法来确保不会重复访问已经生成网络结构的那些节点。从技术的角度来说，这一点不是特别重要，因为我们可以很容易在最终的边列表中删除重复的行。可是，从实践的角度来说，这很重要。在创建新网络结构之前，通过删除候选新种子列表中那些已经访问过的节点，可以减少必须做的API调用次数。这又减少了超过SGA访问频率限制的可能性，而且使脚本的运行时间更短。我们把这个功能加入到twitter.snowball函数中，而不是让get.seeds函数来处理这个问题，这是因为在twitter.snowball函数中已经将整个网络的边列表存储在内存里了。这也确保了我们在建立整个滚雪球取样的过程中，内存里没有存放整个网络的边列表的多份拷贝。请记住：在这种数据规模下，我们需要非常注意内存，特别是在使用像R这种程序设计语言的时候。

```
twitter.snowball <- function(seed, k=2) {
    # Get the ego-net for the seed user. We will build onto
    # this network to create the full snowball search.
    snowball.el <- twitter.network(seed)

    # Use neighbors as seeds in the next round of the snowball
    new.seeds <- get.seeds(snowball.el, seed)
    rounds <- 1 # We have now completed the first round of the snowball!

    # A record of all nodes hit, this is done to reduce the amount of
    # API calls done.
    all.nodes <- seed

    # Begin the snowball search...
```

```
    while(rounds < k) {
        next.seeds <- c()
        for(user in new.seeds) {
             # Only get network data if we haven't already visited this node
             if(!user %in% all.nodes) {
                user.el <- twitter.network(user)
                if(dim(user.el)[2] > 0) {
                    snowball.el <- rbind(snowball.el, user.el)
                    next.seeds <- c(next.seeds, get.seeds(user.el, user))
                    all.nodes <- c(all.nodes, user)
                }
             }
        }
        new.seeds <- unique(next.seeds)
        new.seeds <- new.seeds[!which(new.seeds %in% all.nodes)]
        rounds <- rounds + 1
    }
    # It is likely that this process has created duplicate rows.
    # As a matter of housekeeping we will remove them because
    # the true Twitter social graph does not contain parallel edges.
    snowball.el <- snowball.el[!duplicated(snowball.el),]
    return(graph.edgelist(snowball.el))
}
```

twitter.snowball函数实现的功能正是你所期望的。这个函数有两个参数：seed和k。参数seed是表示Twitter用户的字符串，在我们的案例中，它应该是“drewconway”或者“johnmyleswhite”。参数k是一个大于等于2的整数，它决定滚雪球取样会进行几轮搜索。在网络常用语中，*k*一般是指一个图中的度或者距离。在这个案例中，它是指滚雪球取样将会访问到的节点在网络中与种子节点的最大距离。我们第一步是由种子节点建立初始的个体网络，并以此为起点，获得新的种子节点开始下一轮。

我们通过调用twitter.network和get.seeds来完成上述工作。现在就可以开始迭代地进行滚雪球取样了。整个取样过程发生在twitter.snowball函数的while循环中。循环基本逻辑结构如下：首先，判断我们是否达到取样最大距离。如果没有，就继续进行下一层的滚雪球取样。我们通过迭代地为每个新种子用户建立个体网络来完成取样任务。all.nodes对象用于保存已经访问过的节点，换句话说，在建立一个节点的个体网络之前，我们先检查它是否在all.nodes对象中。如果不在，那么把这个用户的关系增加到snowball.el、next.seed和all.nodes对象中。在进行下一轮的滚雪球取样之前，我们检查新的new.seeds向量，确保没有任何的重复元素。正如前面所说的，这样做是为了防止重复访问节点。最后，我们增加轮次计数器。

在最后一步中，我们获得snowball.el矩阵，它表示整个滚雪球取样获得的样本边列表，然后使用graph.edgelist函数把它转换成一个igraph图对象。可是，在转换之前，我们最后还有一件必须要做的事。因为SGA中的数据并不完美，偶尔有一些样本之间包含重复的关系。从图模型理论上来说，我们可以保留这些关系，并且使用一种叫做“多重

图”（multi-graph）的特殊类型的图来表示我们的Twitter网络。一个多重图只是在节点之间具有多个边的图。尽管这种图也许在某些环境下是有用的，但是对于这个模型来说，几乎没有任何有效的社交网络分析标准方法。另外，因为在这个案例中，那些额外的边是错误造成的，所以我们将在生成一个图对象之前删除这些冗余的边。

我们现在有了建立Twitter网络所需的所有功能性函数。在下一节中，我们将额外编写一个脚本，使用它可以方便迅速地从已知种子用户生成网络结构，并且对生成的那些图做一些基本的结构分析，以发现图中存在的区域群落结构。

分析Twitter社交网络

现在可以开始建立Twitter社交网络了。为了介绍另外一种创建和运行R语言脚本的方法，我们在这个案例中将使用在命令行中运行的shell脚本。到目前为止，我们已经写了很多在R语言控制台中运行的代码，这将是你使用这门语言的主要方式。可是，你偶尔也会碰到需要使用不同输入进行多次运行的任务或者程序。在这种情况下，写一个在命令行运行并且接受标准输入的程序可能会更加方便。我们将使用R语言安装时自带的R脚本命令行程序来完成这个工作。在我们运行程序之前，首先浏览一遍这份代码，以便理解它究竟做了哪些事情。

警告： 回忆一下，因为Google社交关系图API的变化，所以在写下这段话的时候，我们不再能够生成新的Twitter社交网络数据了。因此，对于本章剩余的部分，我们将使用第一次收集到的关于john的Twitter社交网络数据。

因为我们也许想要为很多不同的用户建立Twitter社交网络，所以我们的程序应够能够接受不同的用户名和输入，并且生成相应的数据。在内存中建立了网络对象之后，我们将对它进行一些基本的网络分析，以发现潜在的圈子结构，并把这些信息增加到这个图对象中，然后把它输出到磁盘文件。首先加载igraph函数库和之前建立的函数，这些函数放在*google_sg.R*文件中。

```
library(igraph)

source('google_sg.R')

user <- 'johnmyleswhite'

user.net <- read.graph(paste("data/",user, "/", user, "_net.graphml", sep =""),
format = "graphml")
```

再次使用`igraph`函数库来创建和处理Twitter社交图对象。我们使用`source`函数来加载前面几节所写的函数。因为我们不会去为这个案例研究生成新数据，所以加载了以前保存在*johnmyleswhite*文件夹中的用户John的Twitter网络数据。

警告： 给Windows用户的提示：你也许无法在DOS命令行运行这个脚本。在这种情况下，你只需要把user变量设置成你想要为其建立社交网络的Twitter用户，然后按照你以前正常使用DOS的方式修改并运行脚本。在*twitter_net.R*文件中的代码也需要注意这一点。

在前面几节中创建了所有必要的辅助函数之后，我们只需要把种子用户传给`twitter.snowball`函数，就可以开始工作了。另外，因为在这个例子中我们感兴趣的是建立种子用户的个体网络，所以将只进行两轮滚雪球取样。在本章的后面部分，我们将把网络文件加载进图形可视化软件Gephi中，它允许我们为数据建立漂亮的可视化图形。Gephi有一些用于可视化的保留变量名。其中之一是`Label`变量，它用来标记图中的节点。因为`igraph`默认把标记信息存放在叫做name的顶点属性中，所以我们需要创建一个叫做Label的新顶点属性，它的值和name一样。我们使用`set.vertex.attribute`函数来完成这个工作，它接受上一步创建的`user.net`图对象、一个新的属性和为这个属性赋值的数据向量作为输入。

```
user.net <- set.vertex.attribute(user.net, "Label",
    value = get.vertex.attribute(user.net, "name"))
```

注意： Gephi是一个开源、跨平台图形可视化与操作平台。虽然R语言的内置网络可视化工具很有用，但是并不满足我们的需求。Gephi是特意为网络可视化设计的，并且包含了许多用于创建高质量网络图形的有用特性。如果你没有安装Gephi或者已经安装了一个旧版本。我们强烈推荐你安装最新的版本，可以从*http://gephi.org/*获得。

现在已经建立了图对象，我们将对它做一些基本网络分析，以降低复杂性并且发现一些它的区域群落结构。

区域圈子结构

分析过程中的第一步是提取图的核心元素。我们想要从`user.net`对象中提取两个有用的子图。我们将进行“*k*核分析”来提取图形的2核子图——这是第一个子图。根据定义，*k*核分析将基于节点的连通性来分解一个图。为了找到一个图的“核数”，我们想要知道具有某个度的节点有多少个。*k*核分析中的k表示分解所得子图的度。因此，一个图的2核子图就是由度大于等于2的节点构成的子图。我们对于提取2核子图感兴趣的原因是，进行滚雪球搜索的一个副作用是会在网络外围产生很多单边连接的附属节点。因为这些附属节点对网络结构信息的贡献非常少，所以要删除这些点。

可是，Twitter图是有向的，也就是说它的节点既有入度又有出度。为了找到与分析目标最相关的那些节点，我们将使用`graph.coreness`函数来计算每个节点的核数，通过将函数参数设置为`mode="in"`，来使用节点的入度进行计算。这样做是因为我们想保留那些

至少接收两个边的节点，而不是那些至少发出两个边的节点。由于那些在滚雪球取样中要删除的节点很可能具有指向网络的连接，而不是来自网络的连接，因此我们使用入度来找到它们。

```
user.cores <- graph.coreness(user.net, mode="in")
user.clean <- subgraph(user.net, which(user.cores>1)-1)
```

subgraph函数接收一个图对象和一个节点集作为输入，并且返回原图由这些节点产生的子图。为了提取user.net图对象的2核子图，可以使用R语言的which函数来找到那些入度核数大于1的节点。

```
user.ego <- subgraph(user.net, c(0, neighbors(user.net, user, mode = "out")))
```

警告：使用igraph进行工作时，最令人沮丧的问题之一是igraph使用0索引存储节点，而R语言的索引从1开始。在2核子图的例子中，你将会注意到我们从which函数的返回结果中减去1，以免陷入可怕的“差1”错误。

我们将要提取的第二个关键子图是种子节点的个体网络。回忆一下，它是由种子节点的邻居产生的子图。幸运的是，igraph中的neighbors函数可以识别这些节点。可是和前面一样，我们必须考虑到Twitter的有向图结构。因为一个节点的邻居既可以是入边连接的，也可以是出边连接的，所以我们必须告诉neighbors函数想要哪一种。对于这个例子来说，我们将使用由出度邻居产生的个体网络，换句话说就是种子用户关注的那些Twitter用户，而不是那些关注种子的用户。从分析的角度来说，研究某个人关注的那些用户也许更加有趣，特别是在你研究与自己相关的数据时。这也就是说，另外一种研究方式可能也是有趣的，而我们也欢迎读者使用入度邻居重新运行这份代码，来看看会有什么结果。

在本章的剩余部分，我们将专注于个体网络，但是2核子图对其他网络分析很有帮助，因此将把它作为数据提取结果的一部分进行保存。现在，我们已经做好准备来分析个体网络以找到区域圈子结构了。在这个例子中，我们将使用最基本的一个方法来决定圈子成员：基于节点距离进行层次聚类。这个方法很难用一句话说清楚，但是它的概念相当简单。假设节点（在这个案例中就是Twitter用户）连接的越近（它们之间的中间节点个数越少），就越相似。这是非常合理的，因为我们认为具有共同兴趣的人们会在Twitter上互相关注，那么他们之间的距离就更短。因为在人们的个体网络中也许存在多个圈子，所以已知一个用户，一件很有趣的事情是观察他的圈子网络结构是什么样子的。

```
user.sp <- shortest.paths(user.ego)
user.hc <- hclust(dist(user.sp))
```

进行这个分析的第一步是测量图中所有节点之间的距离。我们使用shortest.paths函数

进行计算，它返回一个N × N矩阵，在这里N是图中节点的个数，而每一对节点之间的距离就是矩阵中由这两个节点定位的元素值。我们将使用这些距离来基于节点之间的邻近程度计算节点分割。顾名思义，“层次”（hierarchical）聚类有很多的层。随着分割个数的增加，聚类过程试图将距离最近的节点保持在相同分割中，层次聚类中的层（也就是分割）就是这样被创建出来的。扩展层次结构每增加一层，我们都把分割数加1。通过这个方法，我们可以把一个图迭代地分解为更小粒度的节点组，一开始所有的节点都在同一个组中，并且不断向下进行层次分解，直到所有的节点自身都构成一个单独的组。

R语言自带了许多方便的函数用于聚类。在这个例子中，我们将使用`dist`函数和`hclust`函数的组合。`dist`函数将由观察矩阵生成一个距离矩阵。在这个案例中，因为已经用`shortest.paths`函数计算出了距离，所以在这里`dist`函数仅仅是把已经计算出的矩阵转换成`hclust`函数可以使用的格式。而`hclust`函数的功能是进行聚类，并且返回一个包含我们所需全部聚类信息的特殊对象。

当你已经有了某些层次聚类的结果之后，一件有意义的事情是观察聚类分割的树状图。一个树状图就是像树那样的图形，它展示了随着层次从上向下移动，聚类如何进行分割。这样做将使我们对个体网络的圈子结构有一个初步了解。例如，我们来检查John的Twitter个体网络树状图，它就存放在本章的`data`文件夹中。为了查看他的树状图，我们将加载他的个体网络数据并进行聚类，然后把`hclust`函数对象传给`polt`函数，它知道怎么画出这个树状图。

```
user.ego <- read.graph('data/johnmyleswhite/johnmyleswhite_ego.graphml', format =
                       'graphml')

user.sp <- shortest.paths(user.ego)
user.hc <- hclust(dist(user.sp))

plot(user.hc)
```

观察图11-3，可以看到John的Twitter社交网络出现了某些确实很有趣的圈子结构。在两个高层次圈子之间，看上去分割得相当明显，而它们内部则是更小的圈子，并且结合非常紧密。当然，每个人的数据都有所不同，而你能看到的圈子个数将很大程度上取决于你的Twitter个体网络的大小和密度。

尽管从学术研究的角度来说，使用树状图检查聚类是很有意义的，但是我们更想用网络视图观察它们。为了做到这一点，需要把聚类分割数据增加到节点中，可以通过一个简单的循环把前10个非平凡的分割增加到网络中。“非平凡”的意思是指，除了第一个分割之外的分割，因为那个分割把所有的节点放到相同的组中。尽管就整个聚类层次而言它很重要，但是它不会带给我们任何区域圈子结构。

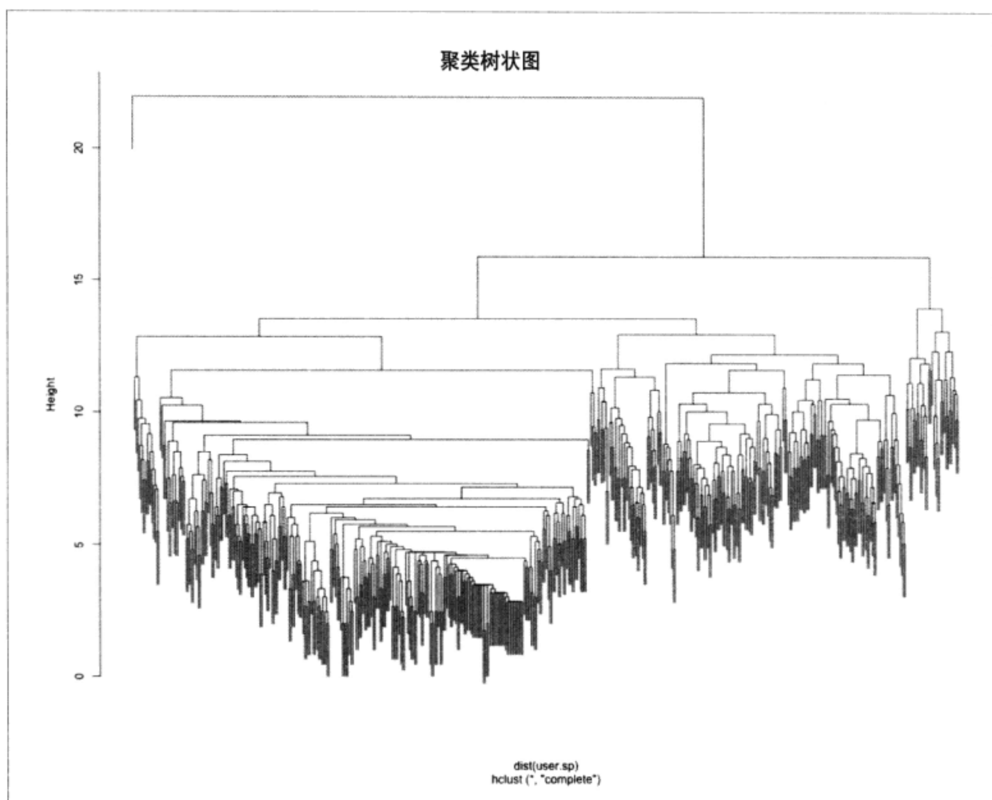


图11-3：John的Twitter个体网络层次聚类树状图，沿着y轴进行层次分割

```
for(i in 2:10) {
    user.cluster <- as.character(cutree(user.hc, k=i))
    user.cluster[1] <- "0"
    user.ego <- set.vertex.attribute(user.ego, name=paste("HC",i,sep=""),
        value=user.cluster)
}
```

cutree函数在层次结构中为给定层的每个节点选择所属分割（即它完成了树的切割）。聚类算法并不知道我们把以种子节点为焦点的个体网络传给了它，所以它把种子节点和其他节点一起聚类到层次结构的每一层。为了在后面的可视化过程中更容易识别种子用户，我们把它分配给它自己的聚类："0"。最后，我们使用set.vertex.attributes函数把这些信息增加到图对象中。现在我们已经有了一个包含Twitter用户名，以及分析得到的前10个聚类分割的图对象。

在可以使用Gephi对结果进行可视化之前，我们需要把这个图对象保存到文件中。我们将使用write.graph函数把这些数据导出为GraphML文件。GraphML是众多的网络文件

存储结构之一，而且是基于XML的。它对我们很有用，因为我们的图对象包含了许多元数据，比如节点标签和聚类分割。和大多数基于XML的格式一样，GraphML文件大小可能会迅速膨胀，并且也不适合简单地存放关系数据。关于GraphML的更多信息，请参考*http://graphml.graphdrawing.org/*。

```
write.graph(user.net, paste("data/", user, "/", user, "_net.graphml", sep=""),
    format="graphml")
write.graph(user.clean, paste("data/", user, "/", user, "_clean.graphml", sep=""),
    format="graphml")
write.graph(user.ego, paste("data/", user, "/", user, "_ego.graphml", sep=""),
    format="graphml")
```

我们保存了这个过程中产生的三个图对象。在下一节中，我们将使用本章的示例数据，通过Gephi对这些结果进行可视化。

使用Gephi可视化Twitter聚类网络

前面提到过，我们可以使用Gephi程序可视化地研究网络数据。如果你已经下载并安装了Gephi，首先要做的是打开应用程序并且加载Twitter数据。在这里例子中，我们将使用Drew的个体网络，它存放在本章的*code/data/drewconway/*文件夹下。不过，如果你生成了自己的Twitter数据，我们欢迎你用它来替换Drew的数据。

注意： 我们并不打算在这里完整而详细地介绍如何使用Gephi进行网络可视化。这一节只是想要说明如何可视化地分析Twitter个体网络的区域圈子结构。Gephi是一个用于网络可视化的健壮程序，它包含了许多用于分析数据的操作选项。在本节中，我们将使用这些操作的一小部分，但是强烈建议你尝试一下这个程序，并且多试试它的各种操作。Gephi自己的快速入门介绍是一个很棒的起点，可以访问：*http://gephi.org/2010/quick-start-tutorial/*。

打开Gephi之后，你可以使用菜单栏中的File→Open（文件→打开）选项加载个体网络。像图11-4上面部分展示的那样，先定位到derwconway文件夹，然后打开其中的*drewconway_ego.graphml*文件。当你加载了图数据之后，Gephi将会返回刚刚被你加载的网络文件的一些基本报告信息。图11-4的下面部分展示了这个报告信息，它包含了节点数和边的个数等信息。如果点击报告窗口上的报告标签，你将会看到我们添加到这个网络上的所有属性数据。值得特别注意的是，以HC开头的那些节点属性，它们是前10个非平凡层次聚类分割的标签。

首先注意到的是，Gephi刚刚加载的网络就像是一大堆随机摆放的节点，是一个可怕的网络毛线团。通过精巧地摆放节点位置，可以表达网络中的许多群落结构。摆放大型复杂网络节点的方法和算法是计算机行业中经常用到的，因此，你可以用很多方式来重新摆放这些节点。对于我们来说，希望那些有更多共同连接的点摆放得更靠近。回忆一

下，我们的聚类方法就是基于节点之间的距离把节点放到不同组中。距离更近的节点会被放在同一组中，而且我们希望可视化技术能够反映这一点。

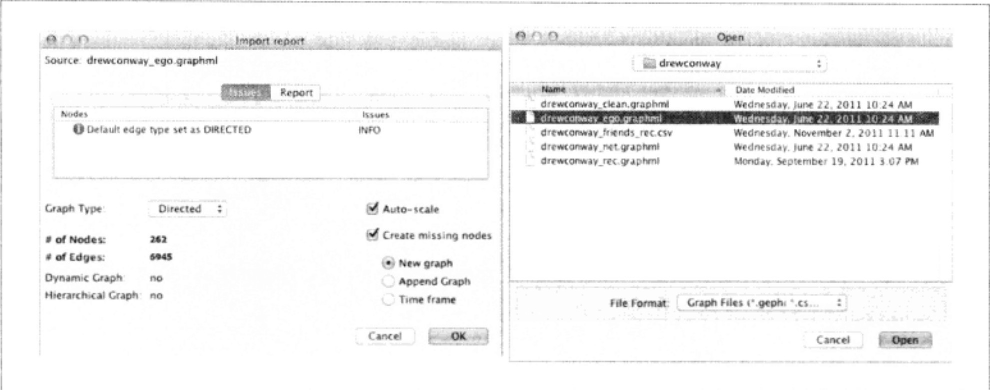


图11-4：加载网络数据到Gephi：a) 打开网络文件 b) 数据加载进Gephi

一组常用的节点摆放方法组成了“力导向”（force-directed）算法。顾名思义，这些算法试图模拟：如果把引力和斥力放到网络中，如何对这些节点进行摆放。把Gephi当前展示的图中的那些节点间混乱的边想象成橡皮筋，并且把节点想象成具有相同磁性的球形轴承。力导向算法试图计算球形轴承因为磁力排斥被彼此推开多远，但是随后被橡皮筋拉回来多少。这个算法的可视化结果是，节点按照区域圈子结构整齐地摆放到一起。

Gephi自带了许多力导向布局样例。在布局（Layout）面板中，下拉菜单中有许多不同的选项，其中一些就是基于力导向的。我们将选择胡一帆比例（Yifan Hu Proportional）算法，并且使用默认设置。选择这个算法之后，点击运行（Run）按钮，你将看到Gephi使用力导向的方式重新摆放节点。这个过程花费的时间依赖于你的网络大小，而且特别依赖你运行时使用的硬件设备。当节点停止移动时，算法就完成了对节点摆放位置的优化，我们也做好了进行下一步工作的准备。

为了更容易识别网络中的区域圈子和它们的成员，我们将改变节点的大小并且对它们重新着色。因为这里的网络是有向个体网络，所以我们将把节点的大小设置为一个关于节点入度的函数。这将使种子节点成为最大的节点，因为几乎每一个网络成员都关注种子，这样也将会使个体网中其他突出的用户节点增大。在排序（Ranking）面板中点击节点（Nodes）标签，并且在下拉框中选择入度（InDegree）。点击红色钻石图标，可以设置大小。你能够把最大尺寸和最小尺寸设置成任何你想要的值。你可以在图11-5的下半部分中看到，我们在Drew的网络中把最小尺寸和最大尺寸分别设置为2和16，但是其他设置也许对你来说会更好。当你设置了这些值之后，点击应用（Apply）按钮，即可改变节点的大小。

最后一步是根据不同的圈子分割给节点着色。在排序面板前面的分割（Partition）面板中，你将看到一个有两个相对箭头的图标。点击这个图标，就可以刷新这个图的分割列表。在你完成这些之后，旁边的下拉框中将包含我们为这些分割引入的节点属性数据。如图11-5的上半部分所示，我们选择了HC8，或者说第8个分割，它包含了一个由Drew及其个体网络中其他7种节点组成的分割。再次点击应用按钮，这些节点将会按照分割进行着色。

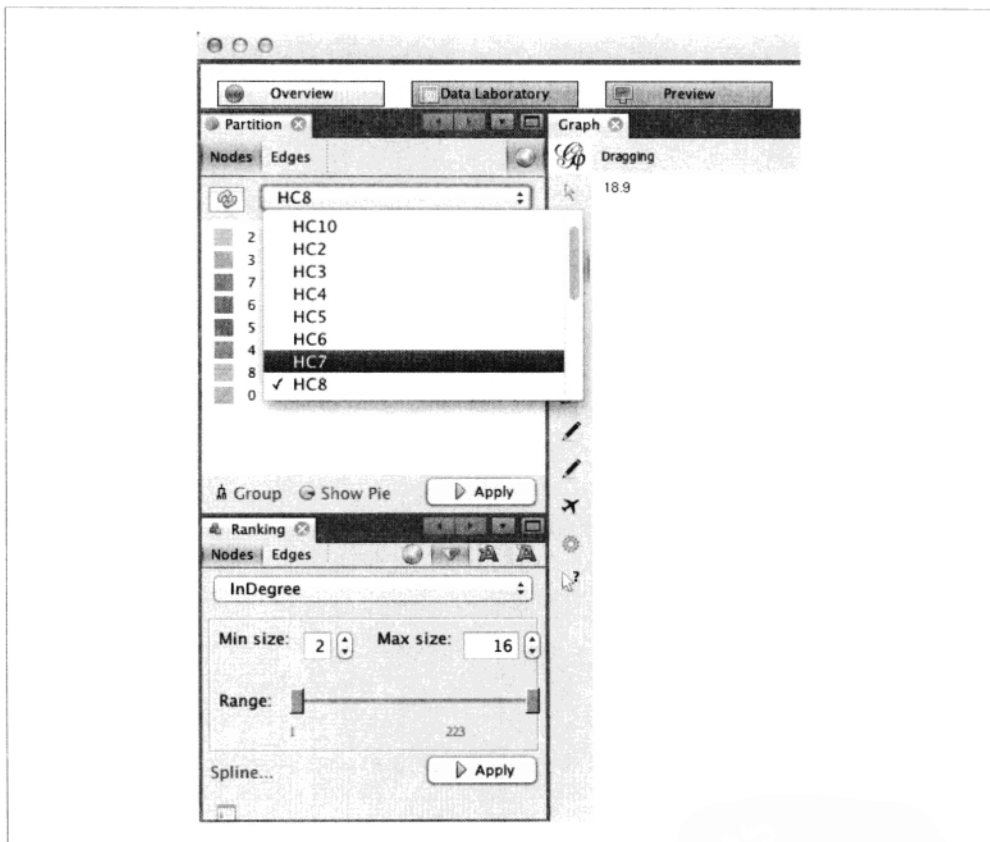


图11-5：通过聚类设置节点的大小并着色

你将立刻看到网络中潜在的结构！用于观察一个特定网络如何分裂成更小的圈子的一个完美方法是，一步一步地对一个层次聚类的分割进行这样的操作。作为一个练习，我们建议你在Gephi中通过增加分割粒度来迭代地对节点进行重新着色。从HC2到HC10，每一次都对节点重新着色，然后观察网络被分成多大的组。这样做可以告诉你网络的很多潜在结构。图11-6展示了Drew的个体网路如何基于HC8被着色，它漂亮地突出显示了个体网络的区域圈子结构。

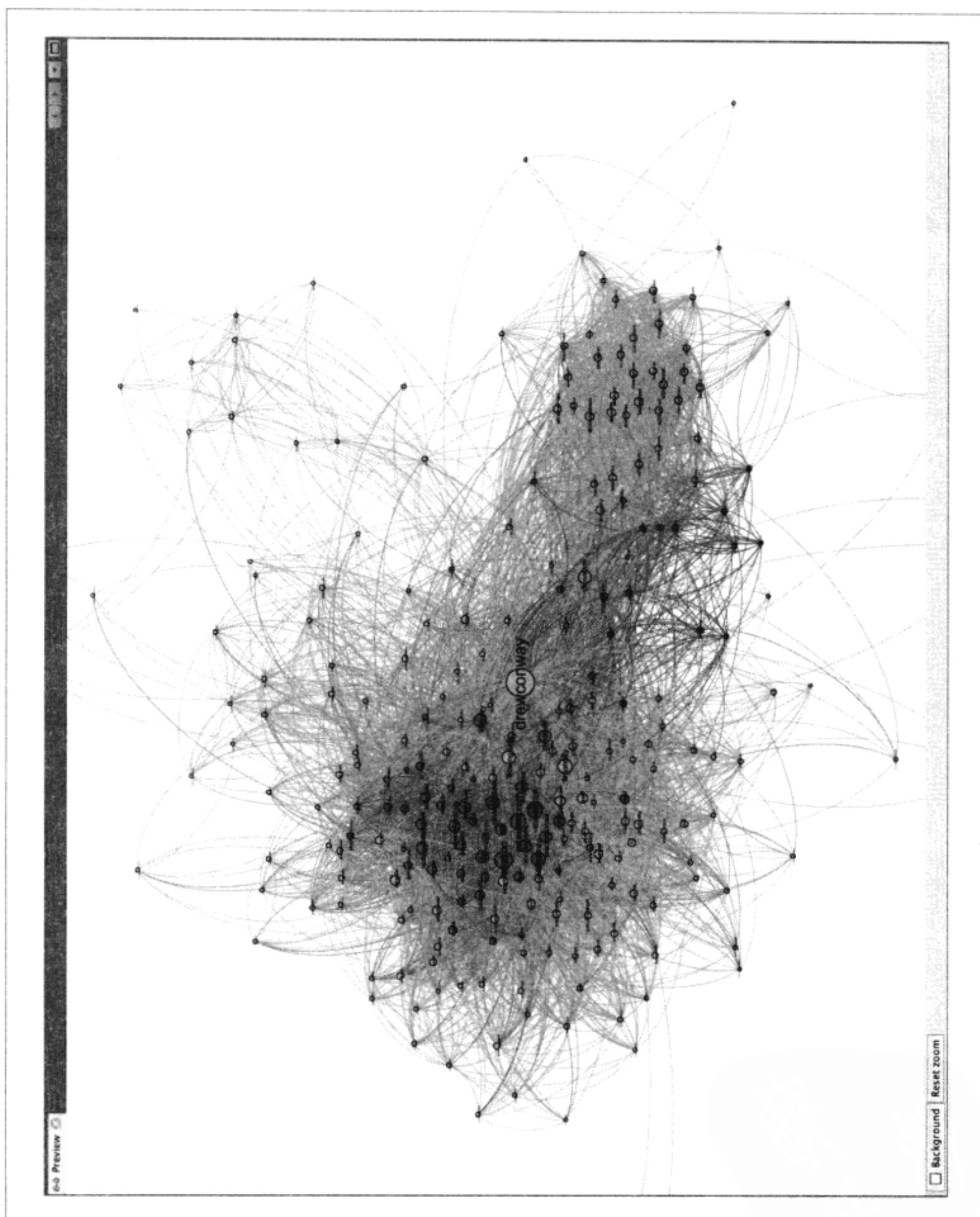


图11-6：根据区域圈子结构对Drew的个体网络着色

看上去Drew的个体网络有四个主要圈子。随着中间代表Drew的节点被着色为青色，我们可以看到两个与它紧密相连的组，分别着色为红色和紫色；还有另外两个与他连接不

是那么紧密的子组，分别是他右边的蓝色组和绿色组。当然，还有其他橘黄色、粉红色和浅绿色的组等，但是我们只关注4个主要的组。

在图11-7中，我们专注于网络的左半部分，并且把边都删除了，以便于更容易观察节点的标签。快速浏览一遍这部分聚类的Twitter用户名，很明显Drew的这部分网络包含了Drew在Twitter上关注的数据专家。首先我们看到的是知名的数据专家，比如浅绿色的蒂姆·奥莱利（timoreilly）和Nathan Yau（flowingdata），因为他们都是自成一体的。紫色和红色的组也很有趣，因为它们都含有数据黑客，但是被一个关键因子分成两部分：Drew的紫色好友都是数据圈子的杰出成员，例如：HilaryMason（hmason）、Pete Skomoroch（peteskomoroch）、Jake Hofman（jakehofman），但是他们没有一位是R语言圈子的活跃成员。另一方面，红色的节点都是R语言圈子的活跃成员，包括Hadley Wickham（hadleywickham）、David Smith（revodavid）、Gary King（kinggary）。

此外，力导向算法成功地把这些圈子成员放到一起，并且把属于这两个圈子的那些节点放到圈子的边缘。我们可以看到John（johnmyleswhite）是紫色的，但是他被放到很多红色节点中间。这是因为John在这两个圈子中都是杰出成员，而且数据也反映了这一点。其他的这类例子包括：JD Long（cmastication）和Josh Reich（i2pi）。

尽管Drew花了很长时间和数据圈子成员交流（包括R用户和非R用户数据圈子成员），但是Drew也使用Twitter与满足其他兴趣的圈子交流。其中一个特别的兴趣是他的学术职业生涯，他关注国家安全技术和政策。在图11-8中，我们突出了Drew网络的右半部分，它包含了来自这些兴趣相关的圈子的成员。和数据专家组类似，这部分包含了2个子组，一个是蓝色的，另外一个是绿色的。和前面的例子一样，节点的分割颜色和摆放位置可以反映出他们在网络中扮演的角色。

蓝色分割中的Twitter用户铺得很开：一部分离Drew很近，在网络的左边，而另外一些在网络的右边，接近绿色的组。那些靠近左边的用户与技术在国家安全中的角色这一话题有关，这些用户包括：Sean Gourley（sgourley）、Lewis Shepherd（lewisshepherd）和Jeffrey Carr（Jeffrey Carr）。那些靠近绿色组的用户更加关注国家安全政策，和绿色组中的成员相似。在绿色组中，我们看到很多Twitter上著名的国家安全圈子成员，包括：AndrewExum（abumuqawama）、Joshua Foust（joshua Foust）和Daveed Gartenstein-Ross（daveedgr）。和前面一样，有趣的是，那些属于两个组的人被放置到聚类边缘，例如：Chris Albon（chrisalbon），他在两个圈子中都很杰出。

如果你研究自己的数据，会看到什么样的区域圈子结构呢？也许像Drew的网络一样，圈子结构显而易见，也可能是更加不明显的圈子结构。深入研究这些圈子结构是很有趣的，而且会获得很多信息，我们鼓励你去做这种研究。在接下来的一节中，我们将使用这些圈子结构为推特创建一个我们自己的“感兴趣的人”推荐引擎。

图11-7：Drew的数据专家好友

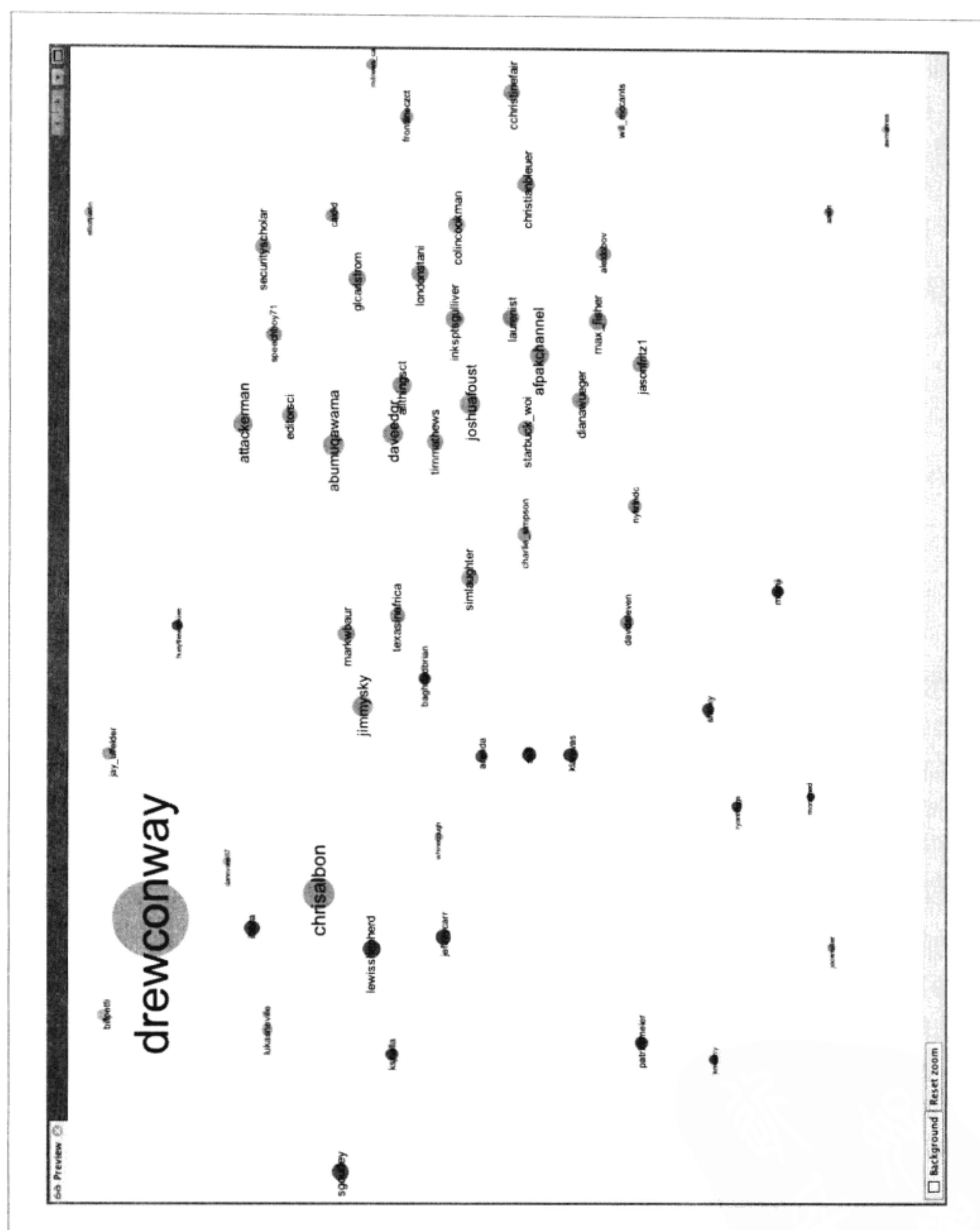


图11-8：Drew的国家安全专家好友

建立“感兴趣的人”引擎

要为Twitter建立我们自己的好友推荐引擎，有很多方法可以考虑。Twitter中的数据有很

多维度，因此我们可以考虑基于用户在Twitter上谈论的话题进行用户推荐。这是文本挖掘的一个实践，而且它要求基于Twitter语料中某些共现的单词或者主题集合来匹配用户。同样，许多tweet中包含了地理定位信息，因此我们也可以为你推荐距离你很近的活跃用户。或者我们也能把这两种方法综合到一起，两种推荐方法分别取前100个推荐用户，然后计算交集。可是由于本章是关于网络的，因此我们只关注基于用户的关系建立一个推荐引擎。

先让我们看一个关于关系是如何在大型社交网络中起作用的简单理论，这是一个很好的起点。弗里茨·海德在1958年提出了“社会平衡理论”（social balance theory）的概念。

> 朋友的朋友是朋友
>
> 朋友的敌人是敌人
>
> 敌人的朋友是敌人
>
> 敌人的敌人是朋友
>
> ——弗里茨·海德
>
> 人际关系心理学

这个概念非常简单，而且可以在社交关系图中使用闭合和开放的三角关系来描述。海德的理论要求使用有符号的关系，朋友（正）或者敌人（负）。要知道，我们并没有这类信息，那么我们如何使用他的理论来建立一个Twitter关系推荐引擎呢？首先，一个成功的Twitter推荐引擎可以用于把敞开的三角形闭合起来，即找到朋友的朋友，也就是说，找到朋友的朋友，然后把他们作为自己的朋友。尽管我们没有完全符号化的关系，但是如果假设敌人之间在Twitter上互不关注，我就能知道所有的符号为正的关系。

在进行初始数据提取时，我们进行了两轮滚雪球搜索。这份数据包含我们的朋友以及朋友的朋友。因此，我们可以使用这份数据标识需要闭合的三角形。问题在于：在众多潜在的三角形中，我应该推荐首先闭合哪一个呢？我们可以再看一下社会平衡理论。通过寻找滚雪球搜索初始数据中那些种子没有关注的，但是种子的很多朋友都关注了的节点，我们可以获得用于推荐的候选节点集。这样做对海德的理论做了如下扩充：很多朋友共同的朋友，很可能会成为自己的好朋友。从本质上说，我们想要闭合种子Twitter关系中那些最显而易见的三角形。

从技术的角度上来说，这个方法也比试图用文本挖掘或者地理空间分析来推荐朋友要简单多了。在这里，只需要计算我们朋友的朋友中，谁是被我们的朋友关注最多的。为了完成这个任务，我们首先加载之前收集的整个网络数据。和前面一样，我们将在这个例子中使用Drew的数据，但是如果你有自己的数据，我们建议你使用它。

```
user <- "drewconway"

user.graph <- read.graph(paste("data/", user, "/", user, "_net.graphml",sep=""),
    format="graphml")
```

我们首先获得种子的所有朋友的Twitter用户名。可以使用neithbors函数来获得邻居的向量下标，但是因为igraph的默认下标和R语言不同，所以需要对这些下标值加1。然后，把这些值传给特定的V函数，它将返回图的节点属性，在这个例子中是name属性。接着，使用get.edgelist函数生成图的完整边列表，它是一个很大的N × 2矩阵。

```
friends <- V(user.graph)$name[neighbors(user.graph, user, mode="out")+1]
user.el <- get.edgelist(user.graph)
```

我们现在拥有了所需的全部数据，可以用于计算当前种子用户没有关注的用户分别被多少个种子用户的朋友关注。首先，我们需要识别出user.el中的哪些行包含由种子用户的朋友到当前未被种子用户关注的用户的链接。和前面几章一样，我们将使用向量化的sapply函数，对矩阵的每一行都用一个包含相当复杂测试逻辑的函数做检查。我们想要生成一个包含TRUE和FALSE的向量，用它来决定哪些行包含了种子用户没有关注的“朋友的朋友”。

我们使用ifelse函数来组合这个本身是向量化的检查。检查的开始部分查看本行中的任意元素是否是种子用户，以及第一个元素（起点）不是种子用户的朋友。我们使用any函数组合它们，只要其中之一为真，就跳过这一行。其他的检查条件是看当前行的第二个元素（目标节点）是否是种子用户的朋友。我们关心谁是朋友的朋友，而不是谁关注我们的朋友，因此也忽略这个条件为真的行。依赖于行的个数，这个过程将需要花1～2分钟时间，但是完成这一步后，我们把合适的行提取出来，放到non.friends.el中，并且使用table函数创建包含名字出现次数的表格。

```
non.friends <- sapply(1:nrow(user.el), function(i) ifelse(any(user.el[i,]==user |
    !user.el[i,1] %in% friends) | user.el[i,2] %in% friends, FALSE, TRUE))

non.friends.el <- user.el[which(non.friends==TRUE),]
friends.count <- table(non.friends.el[,2])
```

接下来将要展现结果。我们想找到大部分“明显”的三角形，然后闭合它们，因此我们想要找到这份数据中出现最多的那些用户。我们用table函数创建的向量来创建一个数据框。我们也增加了一个归一化方法（normalized）计算最应该被关注的推荐用户，它计算种子用户的朋友关注候选推荐用户的百分比。在最后一步中，我们按照百分比递减的顺序对数据框排序。

```
friends.followers <- data.frame(list(Twitter.Users=names(friends.count),
    Friends.Following=as.numeric(friends.count)), stringsAsFactors=FALSE)
```

```
friends.followers$Friends.Norm <- friends.followers$Friends.Following/length(friends)
friends.followers <- friends.followers[with(friends.followers, order(-Friends.Norm)),]
```

为了展示结果，通过运行`friend.followers[1:10,]`命令，我们可以观察前10行，或者说推荐的感兴趣人之中最佳的前10个。在Drew的例子中，结果如表11-1所示。

表11-1：社交关系图

Twitter用户	关注他的朋友个数	关注他的朋友百分比
cshirky	80	0.3053435
bigdata	57	0.2175573
fredwilson	57	0.2175573
dangerroom	56	0.2137405
shitmydadsays	55	0.2099237
al3x	53	0.2022901
fivethirtyeight	52	0.1984733
theeconomist	52	0.1984733
zephoria	52	0.1984733
cdixon	51	0.1946565

如果你认识Drew，那么这些名字就会很有意义。Drew的最佳推荐是应该关注Clay Shirky（cshirky），他是纽约大学（NYU）的一位教授，研究技术和互联网在社会中的角色，并出版了很多著作。在我们已经了解Drew两个一般兴趣的情况下，这看上去是一个很好的匹配。记住这一点，剩余的推荐都满足了Drew的一个或者两个一般兴趣。他们是Danger Room（dangerroom）、Wired's National Security blog、Big Data（bigdata）和538（fivethirtyeight），Nate Silver的《New York Times》精选广播博客。当然，还有shitmydadsays[译注1]。

尽管这些推荐很不错，而且从本书刚开始撰写开始，Drew就已经关注了这个引擎推荐的用户，但是也许有更好的方式来推荐用户。因为我们已经知道一个已知种子用户的网络具有很多自然的结构，所以通过这种结构来推荐那些各个组中的用户也许是有效的方法。除了推荐朋友们最关注的朋友，我们也可以推荐那些在某个已知维度上和种子用户类似的朋友的朋友。在Drew的例子中，我们可以推荐他闭合在国家安全圈子、政策圈子、数据圈子或者R语言圈子中存在的开放三角关系。

```
user.ego <- read.graph(paste("data/", user, "/", user, "_ego.graphml", sep=""),
    format="graphml")
friends.partitions <- cbind(V(user.ego)$HC8, V(user.ego)$name)
```

译注1：shitmydadsays是Drew的某个兴趣相关的Twitter。

我们首先需要做的是，重新加载包含分割数据的个体网络。因为我们已经研究过HC8这个分割了，所以将仍然使用它来对推荐引擎做最后的改进。加载网络之后，我们将创建一个friends.partitions矩阵，它每一行的第一列元素是分割编号，而第二列是用户名。对于Drew的数据来说，它是这个样子的：

```
> head(friends.partitions)
     [,1] [,2]
[1,] "0"  "drewconway"
[2,] "2"  "311nyc"
[3,] "2"  "aaronkoblin"
[4,] "3"  "abumuqawama"
[5,] "2"  "acroll"
[6,] "2"  "adamlaiacano"
```

现在我们需要做的就只剩下计算每个圈子中最明显需要闭合的三角关系了。因此，我们定义了partition.follows函数，它接受分割编号作为输入，并且能找到那些我们想要的用户。因为已经计算出了所有的数据，所以这个函数只是简单地查看每个分割中的用户，然后返回被种子用户的朋友关注最多的那个用户。这个函数中出现的唯一错误检查就是if语句，它检查一个已知子集的行数是否小于2。做这个检查的原因是：有一个分割只包含一个用户，就是种子，而且我们也不想从这个手工编造的分割中做出推荐。

```
partition.follows <- function(i) {
    friends.in <- friends.partitions[which(friends.partitions[,1]==i),2]
    partition.non.follow <- non.friends.el[which(!is.na(match(non.friends.el[,1],
        friends.in))),]
    if(nrow(partition.non.follow) < 2) {
        return(c(i, NA))
    }
    else {
        partition.favorite <- table(partition.non.follow[,2])
        partition.favorite <- partition.favorite[order(-partition.favorite)]
        return(c(i,names(partition.favorite)[1]))
    }
}

partition.recs <- t(sapply(unique(friends.partitions[,1]), partition.follows))
partition.recs <- partition.recs[!is.na(partition.recs[,2]) &
    !duplicated(partition.recs[,2]),]
```

现在我们可以通过分割来查看推荐结果了。像之前提到过的，种子的“0”这个分割是没有任何推荐结果的，但是其他的分割有。有趣的是，在某些分割中，我们看到了某些和前面一步推荐结果中相同的用户名，但是大多数都是我们没见过的用户。

```
> partition.recs
  [,1] [,2]
0 "0" NA
2 "2" "cshirky"
3 "3" "jeremyscahill"
```

```
4 "4" "nealrichter"
5 "5" "jasonmorton"
6 "6" "dangerroom"
7 "7" "brendan642"
8 "8" "adrianholovaty"
```

当然，更加令人满意的是在网络中观察这些推荐结果。这样将会更容易看出在哪个圈子中推荐了哪个用户。本章包含的代码将会把这些推荐结果添加到一个新的图文件中，它包含了这些被推荐的节点及其所属的分割编号。我们并不在这里列出代码，这是因为它是家庭作业要完成的重要练习，我们鼓励你从本书的O'Reilly官网下载相关代码，并且仔细阅读它。作为最后一步，我们将对Drew的推荐结果数据进行可视化。结果如图11-9所示。

这些推荐结果相当好！回忆一下，蓝色节点是在Drew的网络中那些介于他的技术和国家安全兴趣之间的Twitter用户。引擎推荐了“Danger Room博客”，它正好满足了Drew两方面的兴趣。绿色节点是那些经常谈论国家安全政策的人，在他们中间，我们的引擎推荐了Jeremy Scahill（jeremyscahill）。Jeremy是《The Nation》杂志的国家安全记者，他完全符合这组人的特点，而且可能也暗示了一些Drew的个人政治观点。

另一方面，红色节点都是R语言群落中的人。推荐引擎建议关注Brendan O'Connor，他是卡耐基·梅隆大学的一位机器学习方向博士生，也是一个经常发表R语言相关tweet的人。最后，紫色的组包含了其他来自数据群落的人。在这组中，引擎的推荐是Jason Morton（Jason Morton），他是宾夕法尼亚州立大学的一位数学与统计学方向的助理教授。所有这些推荐都与Drew的兴趣吻合，但也许这比前面的推荐结果更有价值，因为我们清楚地知道这些人与他的兴趣相投之处。

还有许多其他方法可以用来建立一个推荐引擎，而我们希望你可以动手尝试运行这些代码，并且可以在你的数据上不断地改进它们，以获得更好的推荐结果。

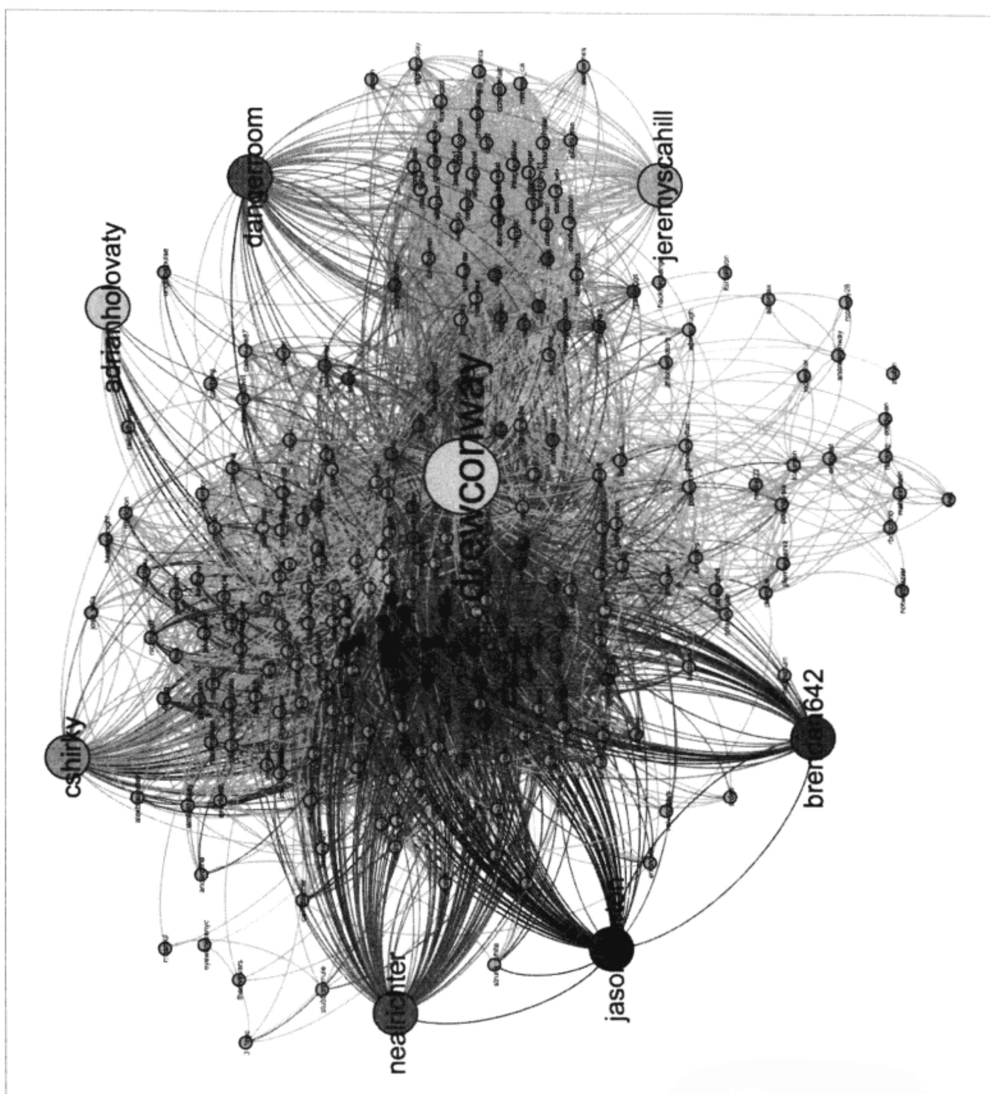


图11-9：Drew的基于区域圈子结构推荐结果

第12章

模型比较

SVM：支持向量机

在第3章，我们引入了决策边界的概念，并指出对于简单的分类算法而言，非线性决策边界的问题会是一个很大的挑战。在第6章，我们演示了如何使用逻辑回归这个线性决策边界分类算法进行分类。在这两章里，我们都承诺会在后面的内容中介绍一种叫做“核方法”的技巧，用来解决非线性决策边界问题。现在，让我们兑现承诺，给你介绍一种新的分类算法——支持向量机（Support Vector Machine， SVM），它可以让你使用多种不同的核函数来找到非线性决策边界。我们会使用SVM来对一组非线性决策边界的数据进行分类。特别是如图12-1所示的数据集。

看一下数据集，我们就很清楚，0类数据分布在两边，而1类数据分布在中间。如果使用在第6章中介绍过的类似逻辑回归这样的简单分类算法，无法找到这种非线性的决策边界。为了证实这一点，我们使用glm函数来尝试使用一下逻辑回归。

```
df <- read.csv('data/df.csv')

logit.fit <- glm(Label ~ X + Y,
                 family = binomial(link = 'logit'),
                 data = df)

logit.predictions <- ifelse(predict(logit.fit) > 0, 1, 0)

mean(with(df, logit.predictions == Label))
#[1] 0.5156
```

正如你看到的，预测准确率只有52%，就算我们猜“所有的数据都是0类数据”，也能得到这样的准确率：

```
mean(with(df, 0 == Label))
#[1] 0.5156
```

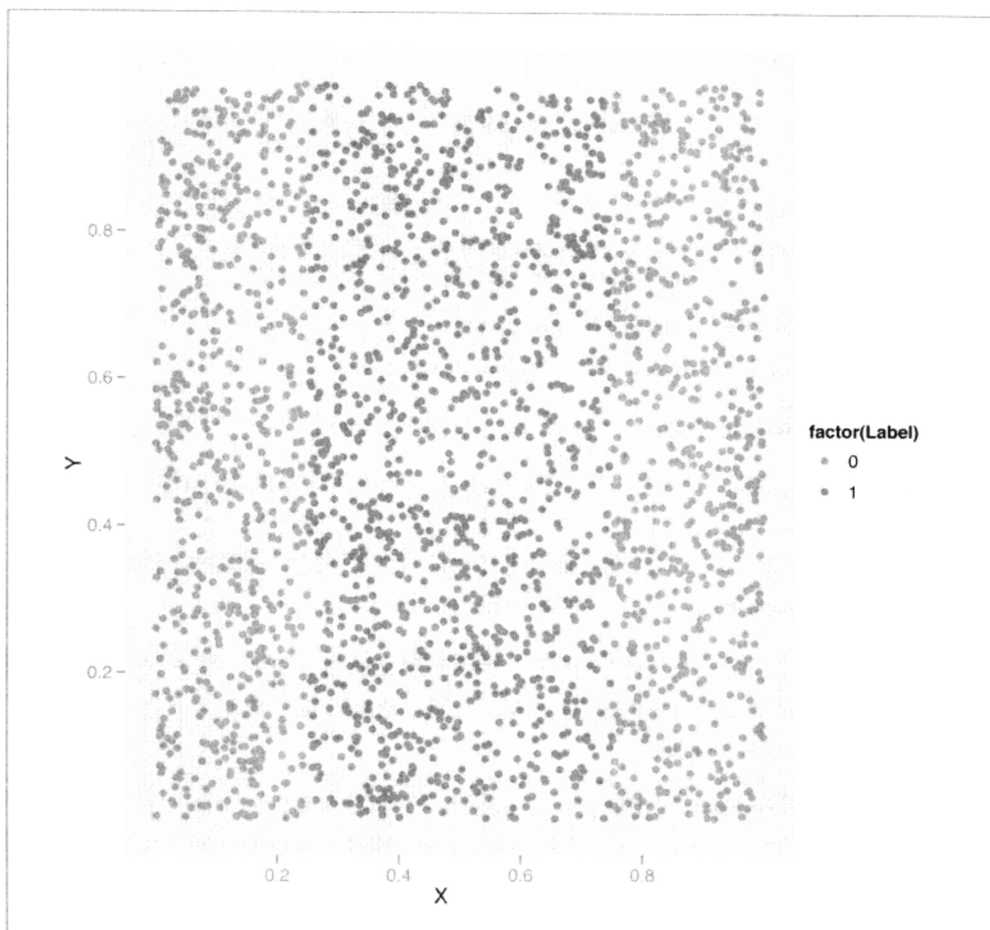


图12-1：非线性决策边界的分类问题

简言之，面对这个问题，逻辑回归模型（包括它所找到的线性分类面）完全没用。它除了利用"0类数据比1类数据要多"这个信息之外，完全没有利用任何其他信息。

那么，我们该怎么办？你将会看到，SVM会干得比逻辑回归更好。我们稍后再解释它具体是怎么实现的，先把整个SVM算法看成一个管用而神奇的黑盒子。我们需要使用程序包e1071中的svm函数，它用起来就像用glm函数一样简单：

```
library('e1071')

svm.fit <- svm(Label ~ X + Y, data = df)
svm.predictions <- ifelse(predict(svm.fit) > 0, 1, 0)
```

```
mean(with(df, svm.predictions == Label))
#[1] 0.7204
```

显然，我们干得好多了（预测准确率为72%），但我们所做的仅仅是把模型从逻辑回归换成了SVM。SVM是怎么做到这一切的？

为了理解SVM是如何胜过逻辑回归的，我们先把SVM的预测结果点和逻辑回归的预测结果都画在直方图上：

```
df <- cbind(df,
            data.frame(Logit = ifelse(predict(logit.fit) > 0, 1, 0),
                       SVM = ifelse(predict(svm.fit) > 0, 1, 0)))

predictions <- melt(df, id.vars = c('X', 'Y'))

ggplot(predictions, aes(x = X, y = Y, color = factor(value))) +
  geom_point() +
  facet_grid(variable ~ .)
```

在这里，为了方便画图，我们把逻辑回归的预测值和SVM的预测值都添加在原始数据集后面，再使用melt函数构建一个新的数据集，并把这个数据集保存在一个叫做predictions的数据框里面。最后，我们把真实数据、逻辑回归的预测结果和SVM的预测结果三者并排展示在图12-2中，以方便互相比较。

在图12-2中，我们发现，逻辑回归把决策边界画在整个数据集之外，根本没发现真实数据中1类数据在中间呈带状分布的事实。SVM发现了这种带状分布的结构，不过它在两侧靠近边缘的预测结果存在失误。

现在，我们看到SVM的确如承诺的那样，生成了一个非线性的决策边界。不过，它究竟是怎么做到的呢？答案就是核方法（kernel trick）。使用一个数学转换，它把原数据集转移到一个新的数学空间中，在这个新空间里它的决策边界是简单的[译注1]。因为这个数据转换基于一个简单的核函数的计算，所以这种技巧称为核方法。

要理解核方法的数学原理非常困难，不过想得到一些感性认识并不难。SVM函数有一个kernel参数，它可以接受4个值：线性（linear）、多项式（ polynomial）、径向（radial）和S型（sigmoid）。为了理解核函数的工作原理，我们将尝试使用这4个核函数，再把它们的预测结果都画出来（见图12-3）：

译注1：此处是指线性的。

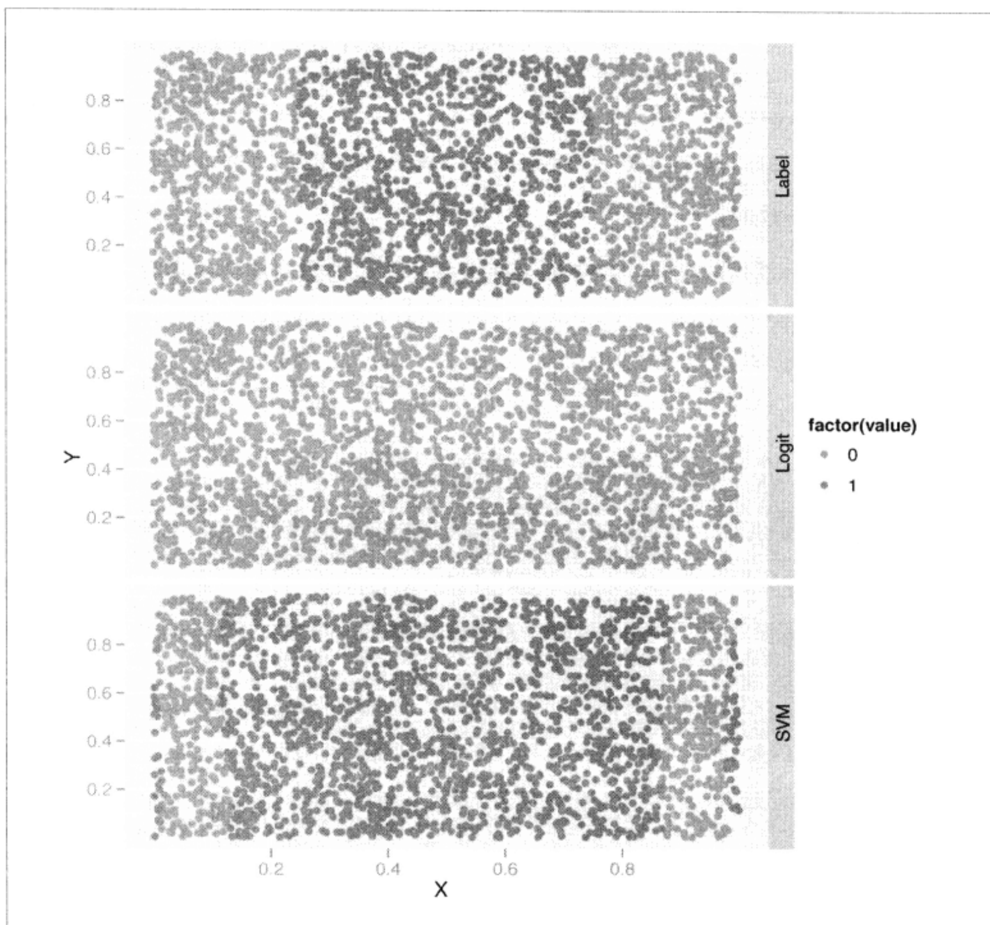


图12-2：对比逻辑回归与SVM的预测结果

```
df <- df[, c('X', 'Y', 'Label')]

linear.svm.fit <- svm(Label ~ X + Y, data = df, kernel = 'linear')
with(df, mean(Label == ifelse(predict(linear.svm.fit) > 0, 1, 0)))

polynomial.svm.fit <- svm(Label ~ X + Y, data = df, kernel = 'polynomial')
with(df, mean(Label == ifelse(predict(polynomial.svm.fit) > 0, 1, 0)))

radial.svm.fit <- svm(Label ~ X + Y, data = df, kernel = 'radial')
with(df, mean(Label == ifelse(predict(radial.svm.fit) > 0, 1, 0)))

sigmoid.svm.fit <- svm(Label ~ X + Y, data = df, kernel = 'sigmoid')
with(df, mean(Label == ifelse(predict(sigmoid.svm.fit) > 0, 1, 0)))

df <- cbind(df,
            data.frame(LinearSVM = ifelse(predict(linear.svm.fit) > 0, 1, 0),
```

```
                        PolynomialSVM = ifelse(predict(polynomial.svm.fit) > 0, 1, 0),
                        RadialSVM = ifelse(predict(radial.svm.fit) > 0, 1, 0),
                        SigmoidSVM = ifelse(predict(sigmoid.svm.fit) > 0, 1, 0)))

predictions <- melt(df, id.vars = c('X', 'Y'))

ggplot(predictions, aes(x = X, y = Y, color = factor(value))) +
  geom_point() +
  facet_grid(variable ~ .)
```

正如你从图12-3中看到的那样，线性和多项式核函数给出的预测结果和逻辑回归差不多。相较而言，径向核函数给出的决策边界就非常接近真实的决策边界。而S型核函数给出的决策边界非常复杂和诡异。

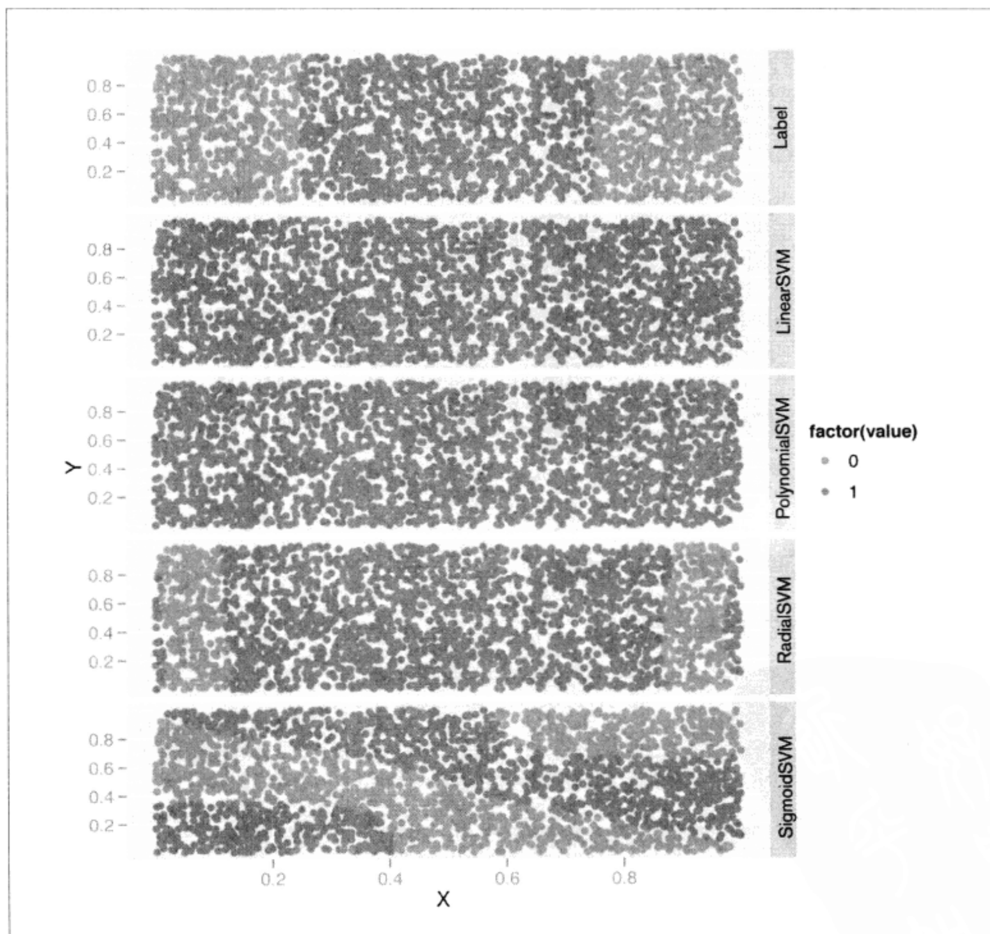


图12-3：SVM不同核函数之间的对比

你应该在自己生成的测试数据集上尝试一下这4个核函数，以便对它们的表现有一个直观的认识。在这之后，你大概会猜想，SVM应该会做得比它看上去能做得更好。的确，SVM有一些默认的超参数需要设置，为了得到最好的预测效果，需要优化这些超参数。接下来我们将介绍最主要的超参数，再看看如何通过优化它们来提升模型的效果。

第一个可以调节的超参数，就是当你使用多项式核函数时的多项式的次数。你可以在调用SVM函数时指定degree参数的值。接下来用4个简单的例子来看一下degree超参数是怎么发挥作用的：

```
polynomial.degree3.svm.fit <- svm(Label ~ X + Y,
                                  data = df,
                                  kernel = 'polynomial',
                                  degree = 3)

with(df, mean(Label != ifelse(predict(polynomial.degree3.svm.fit) > 0, 1, 0)))
#[1] 0.5156

polynomial.degree5.svm.fit <- svm(Label ~ X + Y,
                                 data = df,
                                 kernel = 'polynomial',
                                 degree = 5)
with(df, mean(Label != ifelse(predict(polynomial.degree5.svm.fit) > 0, 1, 0)))
#[1] 0.5156

polynomial.degree10.svm.fit <- svm(Label ~ X + Y,
                                   data = df,
                                   kernel = 'polynomial',
                                   degree = 10)
with(df, mean(Label != ifelse(predict(polynomial.degree10.svm.fit) > 0, 1, 0)))
#[1] 0.4388

polynomial.degree12.svm.fit <- svm(Label ~ X + Y,
                                   data = df,
                                   kernel = 'polynomial',
                                   degree = 12)
with(df, mean(Label != ifelse(predict(polynomial.degree12.svm.fit) > 0, 1, 0)))
#[1] 0.4464
```

在这里，我们发现把degree参数设为3或者5对于模型的预测效果没有太大的影响。（需要指出的是，degree参数的默认值是3）。不过，把degree设到10或者12确实产生了影响。让我们再把决策边界画出来看一下到底发生了什么（见图12-4）：

```
df <- df[, c('X', 'Y', 'Label')]

df <- cbind(df,
            data.frame(Degree3SVM = ifelse(predict(polynomial.degree3.svm.fit) > 0,
                                           1,
                                           0),
                       Degree5SVM = ifelse(predict(polynomial.degree5.svm.fit) > 0,
                                           1,
                                           0),
```

```
                     Degree10SVM = ifelse(predict(polynomial.degree10.svm.fit) > 0,
                                          1,
                                          0),
                     Degree12SVM = ifelse(predict(polynomial.degree12.svm.fit) > 0,
                                          1,
                                          0)))

predictions <- melt(df, id.vars = c('X', 'Y'))

ggplot(predictions, aes(x = X, y = Y, color = factor(value))) +
  geom_point() +
  facet_grid(variable ~ .)
```

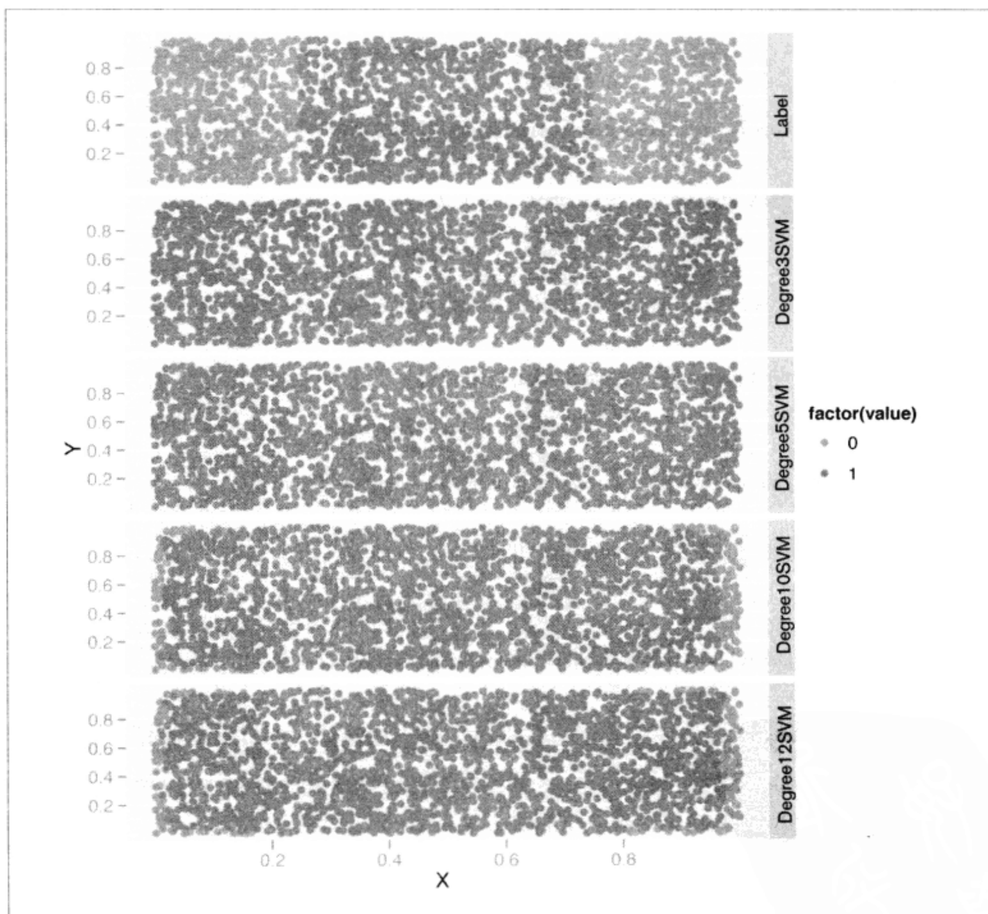


图12-4：不同degree超参数下的多项式核函数SVM

从图12-4中，我们看到更大的degree值确实带来了预测准确率的提升，尽管它是以一种怪异的，并不是真正通过拟合数据的方式达到的。而且，如你注意到的那样，随着

degree越来越大，模型拟合所需要花费的时间越来越长。最后，我们在第6章进行多项式拟合时遇到的过拟合问题，又一次发生了。因此，当你使用多项式核函数来应用SVM算法时，记得一定要对degree使用交叉验证。毫无疑问，如果你肯花精力好好优化它，多项式核函数的SVM分类器会是非常有用的机器学习工具。

我们已经研究过了多项式核函数的degree超参数，接下来研究一下cost超参数，它可以与任何一种SVM核函数相配合。以径向核函数为例，我们尝试4个不同的cost值，来看看改变cost值带来的效果变化。这一次，我们不再数分错类的数据点的个数了，只数一下分类正确的点的个数，毕竟我们还是对模型的"正确性"更感兴趣。下面的代码展示了我们是如何研究这个cost参数的：

```
radial.cost1.svm.fit <- svm(Label ~ X + Y,
                            data = df,
                            kernel = 'radial',
                            cost = 1)
with(df, mean(Label == ifelse(predict(radial.cost1.svm.fit) > 0, 1, 0)))
#[1] 0.7204

radial.cost2.svm.fit <- svm(Label ~ X + Y,
                            data = df,
                            kernel = 'radial',
                            cost = 2)
with(df, mean(Label == ifelse(predict(radial.cost2.svm.fit) > 0, 1, 0)))
#[1] 0.7052

radial.cost3.svm.fit <- svm(Label ~ X + Y,
                            data = df,
                            kernel = 'radial',
                            cost = 3)
with(df, mean(Label == ifelse(predict(radial.cost3.svm.fit) > 0, 1, 0)))
#[1] 0.6996

radial.cost4.svm.fit <- svm(Label ~ X + Y,
                            data = df,
                            kernel = 'radial',
                            cost = 4)
with(df, mean(Label == ifelse(predict(radial.cost4.svm.fit) > 0, 1, 0)))
#[1] 0.694
```

如你所见，增加cost参数使得模型的拟合效果越来越差。这是因为，cost是一个正则化超参数，就像在第6章中描述的lambda参数一样，因此增加cost只会使得模型与训练数据拟合得更差一些。当然，正则化会使你的模型在测试集上的效果更好，因此你最好使用交叉验证来看看什么样的cost值能让你的模型在测试集上取得最好的效果。

为了对拟合的模型有一个直观的认识，让我们看看图12-5中展示的拟合的效果：

```
df <- df[, c('X', 'Y', 'Label')]
```

```
df <- cbind(df,
            data.frame(Cost1SVM = ifelse(predict(radial.cost1.svm.fit) > 0, 1, 0),
            Cost2SVM = ifelse(predict(radial.cost2.svm.fit) > 0, 1, 0),
            Cost3SVM = ifelse(predict(radial.cost3.svm.fit) > 0, 1, 0),
         Cost4SVM = ifelse(predict(radial.cost4.svm.fit) > 0, 1, 0)))

predictions <- melt(df, id.vars = c('X', 'Y'))

ggplot(predictions, aes(x = X, y = Y, color = factor(value))) +
  geom_point() +
  facet_grid(variable ~ .)
```

cost参数的变化带来的改变非常细微，仅可以在图12-5两侧最边缘的数据中察觉到。当cost越来越大，径向核函数的决策边界变得越来越接近线性。

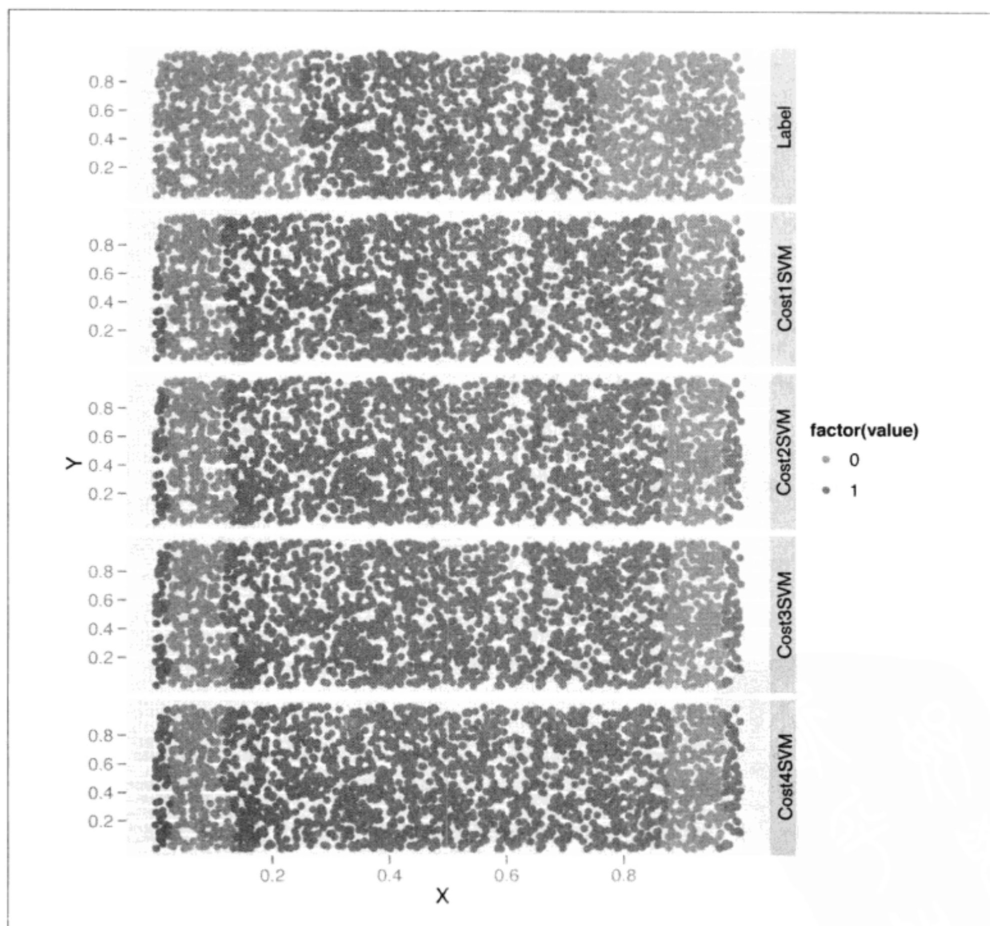


图12-5：不同cost超参数下的径向核函数SVM

探究了cost超参数之后，我们再来研究一下最后一个SVM超参数gamma。这一次，我们将会在S型核函数上测试4个不同的gamma值，来看看它们的表现如何：

```
sigmoid.gamma1.svm.fit <- svm(Label ~ X + Y,
                              data = df,
                              kernel = 'sigmoid',
                              gamma = 1)
with(df, mean(Label == ifelse(predict(sigmoid.gamma1.svm.fit) > 0, 1, 0)))
#[1] 0.478

sigmoid.gamma2.svm.fit <- svm(Label ~ X + Y,
                              data = df,
                              kernel = 'sigmoid',
                              gamma = 2)
with(df, mean(Label == ifelse(predict(sigmoid.gamma2.svm.fit) > 0, 1, 0)))
#[1] 0.4824

sigmoid.gamma3.svm.fit <- svm(Label ~ X + Y,
                              data = df,
                              kernel = 'sigmoid',
                              gamma = 3)
with(df, mean(Label == ifelse(predict(sigmoid.gamma3.svm.fit) > 0, 1, 0)))
#[1] 0.4816

sigmoid.gamma4.svm.fit <- svm(Label ~ X + Y,
                              data = df,
                              kernel = 'sigmoid',
                              gamma = 4)
with(df, mean(Label == ifelse(predict(sigmoid.gamma4.svm.fit) > 0, 1, 0)))
#[1] 0.4824
```

随着gamma的值不断变大，模型变得越来越好。为了对这种进步有一个感性认识，让我们把模型的预测结果画出来（见图12-6）：

```
df <- df[, c('X', 'Y', 'Label')]

df <- cbind(df,
            data.frame(Gamma1SVM = ifelse(predict(sigmoid.gamma1.svm.fit) > 0, 1, 0),
                       Gamma2SVM = ifelse(predict(sigmoid.gamma2.svm.fit) > 0, 1, 0),
                       Gamma3SVM = ifelse(predict(sigmoid.gamma3.svm.fit) > 0, 1, 0),
                       Gamma4SVM = ifelse(predict(sigmoid.gamma4.svm.fit) > 0, 1, 0)))

predictions <- melt(df, id.vars = c('X', 'Y'))

ggplot(predictions, aes(x = X, y = Y, color = factor(value))) +
  geom_point() +
  facet_grid(variable ~ .)
```

正如从图12-6中看到的，当改变gamma时，由S型核函数生成的相当复杂的决策边界会随之弯曲变形。为了有一个更直观的认识，我们建议你使用除了这里4个值之外更多的gamma值来继续这个实验。

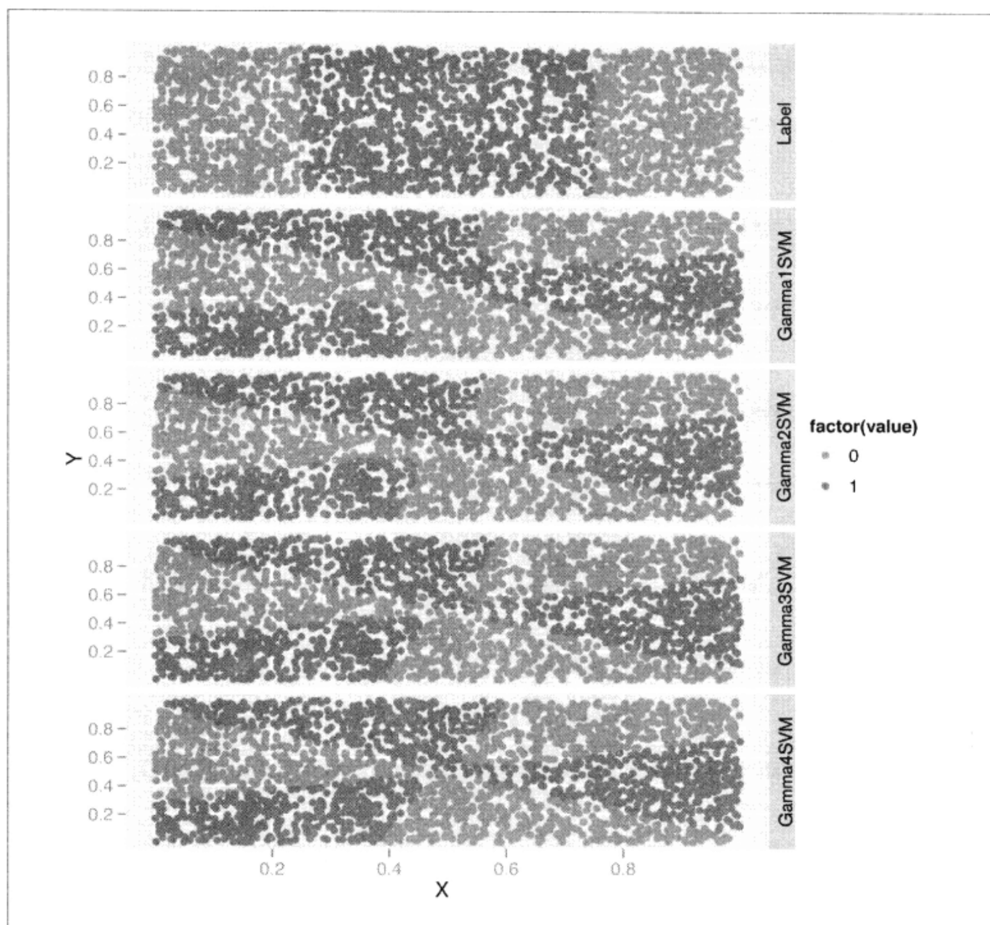


图12-6：不同gamma超参数下的S型核函数SVM

关于SVM的介绍到此为止。我们认为，这会是你机器学习工具箱中非常有用的一个工具。不过现在，你的工具已经足够多了，我们该认真研究一下“当面对一个具体问题的时候，究竟什么工具最合适”这个问题了。针对这个目的，我们将会通过在一个数据集上分别应用多个不同的模型来比较效果。

算法比较

既然我们已经掌握了SVM、逻辑回归和kNN算法，那么就用第3章和第4章中垃圾邮件数据为例子，比较一下它们的表现。面对现实中的一个真实问题，尝试一下多个算法，并比较它们的效果是一个好主意，因为你通常并不知道什么算法在你的数据集上表现最好。而且在机器学习领域，一个菜鸟和一个专家之间的一个重要差别，就在于后者知道

某些特定的问题不适合某些算法。为了培养一个机器学习领域专家那样的直觉，最好的办法就是，对你遇到的每一个机器学习问题，把所有的算法试个遍，直到有一天，你凭直觉就知道某些算法行不通。

第一步是加载数据，然后做一些预处理的工作。因为，这些处理细节之前已经演示过了，因此在此将跳过，直接使用R语言的load函数把文档－词项矩阵从磁盘中加载到内存中，R语言的对象可以以二进制的形式写入磁盘来持久化，而load函数可以把对象从磁盘中重新读取到内存中。接着，我们会把原始数据拆分成训练集和测试集，并使用rm函数把原始数据清理出内存：

```
load('data/dtm.RData')

set.seed(1)

training.indices <- sort(sample(1:nrow(dtm), round(0.5 * nrow(dtm))))
test.indices <- which(! 1:nrow(dtm) %in% training.indices)
train.x <- dtm[training.indices, 3:ncol(dtm)]
train.y <- dtm[training.indices, 1]
test.x <- dtm[test.indices, 3:ncol(dtm)]
test.y <- dtm[test.indices, 1]

rm(dtm)
```

现在数据集已经加载到内存里，我们可以进行下一步了，通过使用glmnet进行一个正则化的逻辑回归拟合：

```
library('glmnet')
regularized.logit.fit <- glmnet(train.x, train.y, family = c('binomial'))
```

当然，还有很多模型调优的工作需要做，让我们尝试一下不同的lambda超参数，看看哪个值能带来最好的表现。为了快速演示这个例子，我们会在交叉验证的过程中有一点的“偷懒”，只在同一份测试集上测试不同的lambda值，而不再对每个不同的lambda值重新进行一次拆分。严格地说，这并不是标准的交叉验证，你不应该学我们这么“偷懒”，不过，我们把标准的交叉验证过程作为练习留给你去完成了。当前，我们只是尝试不同的lambda值，然后看看哪个值在测试集上的表现最好：

```
lambdas <- regularized.logit.fit$lambda
performance <- data.frame()

for (lambda in lambdas)
{
  predictions <- predict(regularized.logit.fit, test.x, s = lambda)
  predictions <- as.numeric(predictions > 0)
  mse <- mean(predictions != test.y)

  performance <- rbind(performance, data.frame(Lambda = lambda, MSE = mse))
}
```

```
ggplot(performance, aes(x = Lambda, y = MSE)) +
  geom_point() +
  scale_x_log10()
```

从图12-7中可以清楚地看到最低错误率对应的lambda值大概在什么范围。为了找到精确的最优lambda值，可以将min函数和取索引运算符结合使用：

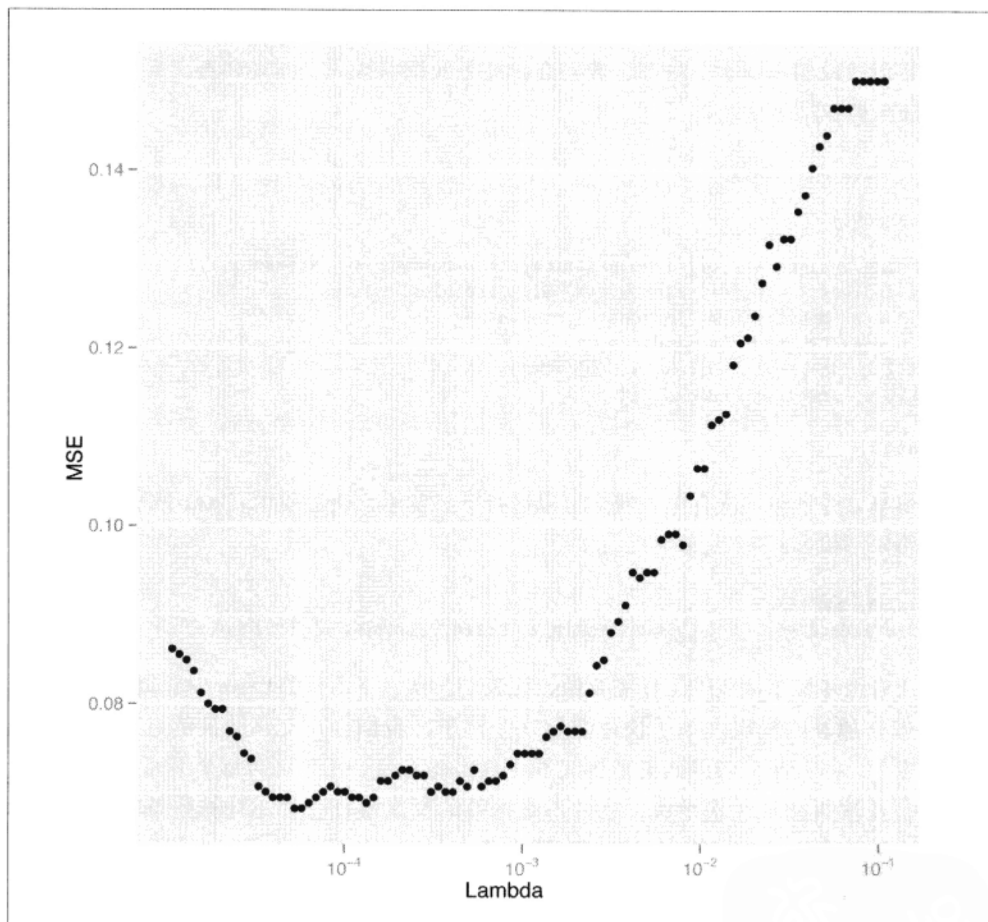


图12-7：找到最优的lambda值

```
best.lambda <- with(performance, max(Lambda[which(MSE == min(MSE))]))
```

其实，在这个的例子中，有两个lambda对应的错误率都是最小的，我们选择了较大的那个lambda，这是因为较大的lambda意味着更强的正则化。我们可以把训练出来的逻辑回归模型在最优lambda下对应的MSE计算出来：

```
mse <- with(subset(performance, Lambda == best.lambda), MSE)
mse
#[1] 0.06830769
```

我们发现，正则化的逻辑回归模型只有一个超参数需要调优，并且错误率只有6%。现在，我们需要看看其他算法的表现，以便决定到底使用逻辑回归、SVM还是kNN。

首先来试一下线性核函数的SVM，看看它和逻辑回归孰优孰劣：

```
library('e1071')
linear.svm.fit <- svm(train.x, train.y, kernel = 'linear')
```

在大数据集上拟合线性核函数的SVM需要花费大量时间。因此，我们打算直接使用默认的超参数，这可能对SVM不太公平。不过，就像我们在前面做逻辑回归的交叉验证时偷懒，因而得到的lambda值不是最优的一样，在这里使用SVM默认的超参数肯定也不是最优的，这是机器学习范畴的一种惯常做法。当你比较模型之间的效果时，有一点必须意识到：你看到的模型一部分的效果，取决于你在这个模型上花费的心血与时间。如果你在一个模型调优上花费的时间比另一个模型多，那么它们之间效果的部分差异，很可能正是因为你为那个模型所花费了更多的调优时间，而不是因为这两个模型本身的优劣。

基于上面的原因，让我们继续评估线性核函数SVM在测试数据集上的效果吧：

```
predictions <- predict(linear.svm.fit, test.x)
predictions <- as.numeric(predictions > 0)
mse <- mean(predictions != test.y)
mse
#0.128
```

我们发现错误率为12%，是逻辑回归模型的两倍。为了获得线性核函数SVM能达到的最优效果，你应该尝试不同的cost超参数。

不过，我们暂时先认为线性核函数的效果就是那样了。面对这个问题，没办法像面对本章开始的那个问题一样有可视化的方式来看不同核函数的SVM效果差异，那么就让我们换一个径向核函数来看看换成核函数能带来多大的改变吧：

```
radial.svm.fit <- svm(train.x, train.y, kernel = 'radial')

predictions <- predict(radial.svm.fit, test.x)
predictions <- as.numeric(predictions > 0)
mse <- mean(predictions != test.y)
mse
#[1] 0.1421538
```

令人意外的是，径向核函数的效果比线性核函数还要差，与本章开始时的非线性数据集上的效果正好相反。从这个例子中，你可以学到一个重要的经验：解决一个问题的最优模型取决于问题数据的内在结构。在本例中，径向核函数表现不好，也许正意味着这个

问题的决策边界可能是线性的。而逻辑回归模型的效果比线性核函数SVM和径向核函数SVM都要好这个事实也支持这个判断。这些观察结果也许是我们在一个固定数据集上比较不同算法所能得出的最有趣的结论了，我们恰恰是从哪些模型效果好，哪些模型效果不好来推测数据的真实结构的。

在认定逻辑回归在这场模型比赛中胜出之前，让我们再来试一下最擅长非线性数据的kNN模型。我们使用50个邻居的kNN算法来进行预测：

```
library('class')
knn.fit <- knn(train.x, test.x, train.y, k = 50)

predictions <- as.numeric(as.character(knn.fit))
mse <- mean(predictions != test.y)
mse
#[1] 0.1396923
```

我们看到，kNN的误差率是14%，这是我们认为线性模型更适合垃圾邮件识别这个问题的又一个有力的证据。因为拟合kNN很快，因此尝试不同k来看下哪个k的效果最好：

```
performance <- data.frame()

for (k in seq(5, 50, by = 5))
{
  knn.fit <- knn(train.x, test.x, train.y, k = k)

  predictions <- as.numeric(as.character(knn.fit))
  mse <- mean(predictions != test.y)

  performance <- rbind(performance, data.frame(K = k, MSE = mse))
}

best.k <- with(performance, K[which(MSE == min(MSE))])
best.mse <- with(subset(performance, K == best.k), MSE)
best.mse
#[1] 0.09169231
```

经过调优之后，kNN的误差率降到了9%。表12-1将我们使用过的各种模型以及它们应用在邮件数据上的误差率列在了一起，kNN的效果介于逻辑回归和SVM之间。

表12-1：模型比较结果

模型	MSE
正则化的逻辑回归	0.06830769
线性核SVM	0.128
径向核SVM	0.1421538
KNN	0.09169231

最后，最优的选择似乎是经过超参数调优的正则化的逻辑回归模型，它最适合我们这个垃圾邮件分类问题。考虑到当前，业界的垃圾邮件分类器的确都在采用逻辑回归模型来淘汰第3章提过的朴素贝叶斯分类器，因此这个结论很合理。尽管原因还不太清楚，但对于这类问题，逻辑回归的效果的确更好。

从这个例子中，我们能学到些什么呢？我们希望你能记住以下几点经验：1）面对实际数据集，你应该尝试多个不同的算法，更何况它们用R语言来实现又那么方便；2）没有所谓通用的“最优”算法，“最优”取决于具体的问题；3）你的模型效果一方面取决于真实的数据结构，另一方面也取决于你为模型的超参数调优所付出的努力，因此，如果你想得到令人信服和满意的结果，多花一点时间在模型的超参数调优上吧。

为了牢记并实际应用这些经验，我们希望你能自己动手，在邮件数据上，重新应用一遍本章中我们使用过的4个模型，重新拆分训练集与测试集，利用交叉验证，对模型的超参数好好调优一番。还有一点，我们在训练SVM过程中没有使用多项式核函数和S型核函数，我们也建议你试一下这两个类型的SVM模型。如果你把我们上面的建议都做了一遍，必定受益匪浅，你还会发现在同样一份数据集上，本书中所提及的模型会有多么大的表现差异。

现在，你终于来到了本章也是本书的最后。我们希望你已经发现了机器学习之美，并且，对于在构建一个数据预测模型时可能遇到的各种各样的问题，有了一定的了解和认识。我们希望你能从那些经典的机器学习教材中继续学习，本书中有些内容，仅仅从实践的目的出发，告诉你“怎么用”，而没告诉你“它是怎么实现的”，以及“它为什么可以”，如果你想从理论角度学习这些内容，我们推荐Hastie 等的著作[HTF09]或Bishop的著作[Bis 06]，最好的机器学习实践者既有实践经验又有理论基础，我们希望你能从两个方面提高自己。

一路上，我们探究机器学习的奥秘，并乐在其中。现在，你有了一大堆强有力的机器学习工具，因此，赶紧找一个你感兴趣的领域，开始机器学习之旅吧！



参考文献

图书

[Adl10] Adler, Joseph. *R in a Nutshell*. O'Reilly Media, 2010.

[Abb92] Abbot, Edwin A. *Flatland: A Romance of Many Dimensions*. Dover Publications, 1992.

[Bis06] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer;1st ed. 2006. Corr.; 2nd printing ed. 2007.

[GH06] Gelman, Andrew, and Jennifer Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2006.

[HTF09] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[JMR09] Jones, Owen, Robert Maillardet, and Andrew Robinson. *Introduction to Scientific Programming and Simulation Using R*. Chapman and Hall, 2009.

[Seg07] Segaran, Toby. *Programming Collective Intelligence: Building Smart Web 2.0Applications*. O'Reilly Media, 2007.

[Spe08] Spector, Phil. *Data Manipulation with R*. Springer, 2008.

[Wic09] Wickham, Hadley. *ggplot2: Elegant Graphics for Data Analysis*. Springer,2009.

[Wil05] Wilkinson, Leland. *The Grammar of Graphics*. Springer, 2005.

[Pea09] Pearl, Judea. *Causality*. Cambridge University Press, 2009.

[WF94] Wasserman, Stanley, and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[MJ10] Jackson, Matthew O. *Social and Economic Networks*. Princeton University Press, 2010.

[EK10] Easley, David, and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning A bout a Highly Connected World*. Cambridge University Press, 2010.

[Wa03] Wasserman, Larry. *All of Statistics*. Springer, 2003.

论文

[LF08] Ligges, Uwe, and John Fox. "R help desk: How can I avoid this loop or make it faster?" *http://www.r-project.org/doc/Rnews/Rnews_2008-1.pdf*. May 2008.

[DA10] Aberdeen, Douglas, Ondrej Pacovsky, and Andrew Slater. "The Learning Behind Gmail Priority Inbox." LCCC: NIPS 2010 Workshop on Learning on Cores,Clusters and Clouds. *http://research.google.com/pubs/archive/36955.pdf*. 2010.

[HW11] Wickham, Hadley. "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software*, April 2011, 40 (1).

[SR08] Stross, Randall. "What Has Driven Women Out of Computer Science?" *The New York Times*, November 15, 2008.

[JC37] Criswell, Joan H.. "Racial Cleavage in Negro-White Groups." *Sociometry*, Jul.-Oct. 1937, 1 (1/2).

[WA10] Galston, William A.. "Can a Polarized American Party System Be 'Healthy'?" *Brookings Institute - Issues in Governance Studies*, April 2010 (34).

[PS08] Singer, Paul. "Members Offered Many Bills but Passed Few." *Roll Call*, December 1, 2008.